

# دلفى ٤ Delphi 4

بايبل  
*Bible*

تأليف  
توم سوان

حقوق الطبع والنشر محفوظة لدار الفاروق للنشر والتوزيع

## دار الفاروق للنشر والتوزيع

أكبر مركز في الشرق الأوسط

لإصدار أحدث الكتب في عالم الكمبيوتر

فرع وسط البلد : ٣ ش منصور المتديان متفرع من شارع مجلس الشعب

محطة مترو سعد زغلول - القاهرة - مصر

تليفون : ٣٥٥٣٠٣٢ / ٠٢ / ٠٢ - ٣٥٤٣٢٠٣ / ٠٢ / ٠٢

فاكس : ٣٥٤٣٦٤٣ / ٠٢ / ٠٢

فرع الدقى : ١٢ ش الدقى - جيزة ت : ٣٣٨١٠٢٢ فاكس : ٨٢٠٧٤

الطبعة العربية الأولى ١٩٩٩

عدد الصفحات : ١٢٢٤ صفحة

رقم الإيداع ٩٩/١١٢٥٧ لسنة ١٩٩٩

الترقيم الدولي 977-307-047-6

الناشر الرئيسى : IDG Books Worldwide. Inc

تاريخ الإصدار للنسخة الإنجليزية الأولى ١٩٩٨

### تحذير

حقوق الطبع والنشر محفوظة لدار الفاروق للنشر والتوزيع  
الوكيل الوحيد لشركة/أى . دى . جى العالمية على مستوى الشرق  
الأوسط ولايجوز نشر أى جزء من هذا الكتاب أو اختزان مادته  
بطريقة الاسترجاع أو نقله على أى نحو أو بأى طريقة سواء كانت  
الكرونية أو ميكانيكية أو بالتصوير أو بالتسجيل أو بخلاف ذلك ومن  
يخالف ذلك يعرض نفسه للمسألة القانونية مع حفظ كافة حقوقنا  
المدنية والجنائية.



*Copyright© 1999 by  
Dar El - Farouk for  
Publishing and Distribution*

Original English language edition copyright © 1998 by **Tom Swan.**

All rights reserved including the right of reproduction in whole or in part in any form.

This edition published by arrangement with the original publisher, **IDG Books Worldwide, Inc.**, Foster City, California, USA."

Delphi is a trademark of Inprise Corporation. The **IDG Books Worldwide** logo is a trademark under exclusive license to **IDG Books Worldwide, Inc.**, from International Data Group, Inc. Used by permission.

شكر وتقدير

تتقدم دار:

الفاروق للنشر والتوزيع

بالشكر لقسم الترجمة بالدار وتخص  
بالذكر كل من:

- د. / خالد العامري

- م/ محمد شمس م/ جيهان مجاهد

على الجهود الكبيرة الذي بذل في ترجمة  
 وإعداد وتنفيذ ومراجعة هذا الكتاب .

## ABOUT IDG BOOKS WORLDWIDE

Welcome to the world of IDG Books Worldwide

IDG Books Worldwide, Inc., is a subsidiary of International Data Group, the world's largest publisher of computer-related information and the leading global provider of information services on information technology. IDG was founded more than 25 years ago and now employs more than 8,900 people worldwide. IDG publishes more than 275 computer publications in over 75 countries (see listing below). More than 90 million people read one or more IDG publications each month.

launched in 1990, IDG Books Worldwide is today the #1 publisher of best-selling computer books in the United States. We are proud to have received eight awards from the Computer Press Association in recognition of editorial excellence and three from Computer Currents' First Annual Readers' Choice Awards. Our best-selling Per Diamond® series has more than 50 million copies in print with translations in 38 languages. IDG Books Worldwide, through a joint venture with IDG's Hi-Tech Beijing, became the first U.S. publisher to publish a computer book in the People's Republic of China. In record time, IDG Books Worldwide has become the first choice for millions of readers around the world who want to learn how to better manage their businesses. Our mission is simple: Every one of our books is designed to bring extra value and skill-building instructions to the reader. Our books are written by experts who understand and care about our readers. The knowledge base of our editorial staff comes from years of experience in publishing, education, and journalism - experience we use to produce books for the '90s. In short, we care about books, so we attract the best people. We devote special attention to details such as audience, interior design, use of icons, and illustrations. And because we use an efficient process of authoring, editing, and desktop publishing our books electronically, we can spend more time ensuring superior content and spend less time on the technicalities of making books.

You can count on our commitment to deliver high-quality books at competitive prices on topics you want to read about. At IDG Books Worldwide, we continue in the IDG tradition of delivering quality for more than 25 years. You'll find no better book on a subject than one from IDG Books Worldwide.



John J. Killebrew  
John Killebrew

John Kilcullen  
CEO  
BXI Books Worldwide, Inc.

Steven Berkowitz

Steven Berkowitz  
President and Publisher  
IDG Books Worldwide, Inc.

[illegible]

## **بيان المسؤولية القانونية**

لقد تم بذل جميع الجهود لتأكيد أن هذا الكتاب يحتوى على معلومات حديثة ودقيقة. وعلى أية حال أى.دى.جى والمؤلف ودار الفاروق للنشر والتوزيع غير مسئولين عن خسارة أو تلف يتكبده القارئ كنتيجة لأية معلومات يحتويها الكتاب.

## **علامة تجارية**

كل العلامات تخص شركاتهم الخاصة.

**اصدار خاص للبيع بالشرق الأوسط**

## محتويات الكتاب

### ★ الجزء الأول: مقدمة

الباب الأول معلومات حول Delphi4

الباب الثاني: معلومات حول الـ Visual Component

الباب الثالث: معلومات حول الـ Forms

### ★ الجزء الثاني: واجهة تطبيق المستخدم

الباب الرابع: برمجة لوحة المفاتيح والفأرة

الباب الخامس: إنشاء القوائم

الباب السادس: Attaching Buttons and Check Boxes

الباب السابع: إنشاء الـ Toolbars، الـ Coolbars، و الـ Status Panels

الباب الثامن: Making Lists

الباب التاسع: العمل مع الـ Single-Line Strings

الباب العاشر: العمل مع الـ Multiple- Line Text

الباب الحادى عشر: الملفات والأدلة

الباب الثانى عشر: الاتصال مع الـ Dialog Boxes

## ★ الجزء الثالث: التطبيق

الباب الثالث عشر: تطوير تطبيقات الـ Graphics

الباب الرابع عشر: تطوير تطبيقات الطباعة

الباب الخامس عشر: تطوير تطبيقات الـ MDI

الباب السادس عشر: التطوير مع الـ Clipboard والـ DDE والـ OLE

الباب السابع عشر: تطوير تطبيقات قاعدة البيانات

الباب الثامن عشر: تطوير الـ Chart والتقارير

## ★ الجزء الرابع: التقنيات المتقدمة

الباب التاسع عشر: Handling Exceptions

الباب العشرين: Constructing Custom Components

الباب الحادي والعشرون: تطوير مهارات Delphi الخاصة بك

## مقدمة

إن Delphi 4 Bible يقدم إرشاداً كاملاً ومرجعاً للبرمجة مع Delphi . إنك لا تحتاج الى خبرة سابقة في مجال البرمجة لكي تستخدم هذا الكتاب ، ولكن اذا كنت تعرف القليل عن ال Pascal أو ال C أو ال C++ أو ال Visual Basic فإن هذا يساعدك الى حد ما . وتقدم تعليمات هذا الكتاب الموضحة خطوة بخطوة تقنيات Delphi من مستوى المبتدئين الى المستويات الأكثر تقدماً . إن لغة ال object Pascal الخاصة بـ Delphi وكذلك ال Visual Components Library موجودة في وثائق ، وسوف تتعلم كيف تنشئ واجهات تطبيق عملية خاصة بالمستخدم للعديد من التطبيقات . والأمثلة الكثيرة - والمتوفرة مع source code - تثير لك طريقك .

### معلومات حول هذا الكتاب:

إن ال Delphi 4 Bible يغطي أوامر ال Delphi ومميزاته وتقنياته في أربعة أجزاء :

• **الجزء الأول: مقدمة:** تقدم الأبواب الموجودة في هذا الجزء البيئة البرمجية الخاصة بـ Delphi وتشرح العديد من أوامره ومميزاته . سوف تتعلم كيفية استخدام ال visual components وال forms والخصائص وال event لإنشاء تطبيقات عملية ذات ٣٢ بت للـ Windows 95 والـ Windows 98 والـ Windows NT .

• **الجزء الثاني: واجهة التطبيق للمستخدم:** توضح أبواب هذا الجزء كيفية إنشاء واجهات تطبيق للمستخدم والـ Visual Components Library الخاصة بـ Delphi . سوف تعرف كيفية برمجة الفأرة ولوحة المفاتيح ، وتنشئ واجهة التطبيق للمستخدم باستخدام ال object مثل القوائم ، الأزرار ، toolbars ، Coolbars ، dockable ، page controls ، dialog box ، scroll bar ، status panels ، windows . سوف تتعلم أيضاً كيف تعمل مع strings أحادي السطر ، Multiple-Line Text ، وتتنقل عبر الأدلة والملفات ، وتصل بـ dialog box .

● **الجزء الثالث: التطبيق:** تغطي أبواب هذا الجزء موضوعات تطوير التطبيق وتشرح كيفية استخدام الـ visual components المتقدمة الخاصة بـ Delphi. سوف تعرف كيفية برمجة تطبيقات للـ Graphics وللصور المتحركة متعددة الوسائط، ومخرجات الطباعة، وتطبيقات MDI، object-linking and embedding وتطوير قاعدة البيانات. سوف تتعلم أيضاً كيف تنشئ تقارير Charts باستخدام الـ visual component QuickReport والـ TeeChart الخاصة بـ Delphi.

● **الجزء الرابع: التقنيات المتقدمة:** تقدم الأبواب الموجودة في هذا الجزء موضوعات شيقة لطوري Delphi المتقدمين سوف تعرف كيف تستخدم الـ exception لمعالجة الأخطاء، وتستخدم وتنشئ المستندات، وتنشئ custom component مخصصة وتحولها الى الـ ActiveX control. والباب النهائي في هذا الجزء (وفي الكتاب يقدم تقنيات برمجة متقدمة للـ Object Pascal. سوف تتعلم الجديد حول الـ object-oriented مثل dynamic arrays، method overloading، default parameters، file streams وعدة أشياء أخرى متنوعة.

**ملحوظة:** ان الجزء الثاني والثالث يقدمان معلومات حول معظم الـ Delphi visual component. لكي تجد مرجعاً للـ component بعينها، ابحث في فهرس الموضوعات في آخر الكتاب.

Note

### متطلبات:

بالإضافة الى هذا الكتاب، إنك تحتاج الى نسخة من طبعات الـ Delphi الـ Professional أو الـ Client/Server الخاصة بـ Delphi تكون مثبتة في حاسبك الشخصي أو الشبكة الخاصة بك. إنك تحتاج ايضاً الى تركيب Windows 95، أو Windows 98، و Windows NT. ان كل التطبيقات والبرامج التعليمية الموجودة في هذا الكتاب تستخدم الـ Visual environment الخاصة بـ Delphi والعاملة في بيئة الـ Windows. بالرغم من ان كثير من المعلومات الموجودة في هذا الكتاب تنطبق على كل نسخ Delphi، الا انه يجب ان تستخدم Version 3 أو Version 4 للحصول على أفضل النتائج.

**Tip** **فكرة:** تأكد من وجود ملفات ال source code الخاصة ب Delphi (غالباً في دليل ال source الخاص ب Delphi). إنك تجد إجابات كثيرة لعدد من التساؤلات بتصفح ال source code ل component ما. إنني أجد قراءة ملف source code لمعرفة مزيد من المعلومات حول component أو موضوع ما.

### الاختلافات بين نسخ ال Delphi:

في Delphi4، إنك تفتح ال Project Options باختيار ال View|Project Manager ثم تضغط يميناً المشروع (1 Project، مثلاً). هذا يعرض قائمة والتي تستطيع ان تختار منها أمر ال Options.... لفتح ال Project Options Dialog. استخدم هذه الطريقة اذا كنت تستخدم Delphi4 عندما يخبرك هذا الكتاب ان تختار ال Project Options.....، ومازال ال Project Options ايضاً متوفراً من خلال ال Project Options menu.

في هذا الكتاب افترض انك تعرف كيف تستخدم ال Windows وانه لديك فأرة أو جهاز شبيه مثل track Ball. وإنني لن اضيع وقتك ووقتي في موضوعات مثل كيفية اختيار اوامر القائمة، ضغط الازرار، والتنقل بين النوافذ. اذا اردت المساعدة في كيفية استخدام ال Windows، عليك باستشارة واحداً من البرامج التعليمية الممتازة الموجودة في السوق أو ارجع الى المراجع والارشادات التي تأتي مع نظام تشغيل ال Windows. إنني ضمنت كثير من الافكار حول الاستخدام الفعال لبيئة التطوير المتكاملة ل Delphi، ولكن لا تنسى ان هذا في الاساس كتاب عن البرمجة. فإن اغلب محتوياته تركز على كيفية كتابة تطبيقات باستخدام Delphi ولغة برمجة ال Object Pascal.

### الاعراف المستخدمة في هذا الكتاب:

ان كل باب يعتبر صغيراً بحيث تستطيع ان تقرأه في جلسة واحدة. واذا انهيت باباً واحداً أو اثنين في اليوم الواحد فإنك سوف تتمكن من اجادة ال Delphi في حوالي أسبوعين. ابق معي لاسبوع آخر وسوف تصبح خبيراً!

وإنني اعترف، رغم ذلك، ان الساعات المطلوبة غير متوفرة في اليوم لكل شخص ليقراً ولو كلمة واحدة في مثل هذا الكتاب الضخم. لكي اسعدك على ان



تجد ما تحتاجه بسرعة- وحتى لا تضيع الوقت فى قراءة ما تعرفه بالفعل - فإن كل الابواب تنتهى بثلاث فصول :

• **مشروعات:** لكى تختبر فهمك لمحتويات الباب ، جرب نفسك فى المشروعات المقترحة فى نهاية كل باب . يحتوى القرص المدمج الخاص بهذا الكتاب يحتوى على أى ملفات مذكورة فى هذا الفصل .

• **افكار للمستخدم الخبير:** تصفح هذه الافكار الإضافية لزيادة فهمك عن موضوع الباب لتتعرف على تقنيات البرمجة .

• **ملخص:** اقرأ هذا الملخص الموجودة فى نقاط لتنشيط ذاكرتك عن محتويات الباب . يمكنك أيضاً تصفح الملخصات لتجد الابواب التى تغطى الموضوعات التى تريدها .

**فكرة:** اذا كنت تعرف ال Pascal بالفعل او اذا كنت تعرف Delphi ، فتصفح اولاً مشروعات ، وافكار ، وملخص كل باب لكى تساعدك على تحديد ما اذا ما كان يجب عليك قراءة الباب ام لا . من خلال نص الكتاب ، فإنك سوف تجد ملحوظات وافكار وتحذيرات اضافية والتى توضح عشرات التقنيات الخاص بـ Delphi . وهذه الفصول - التى لها icons على الجانب الايسر من الصفحة - يمكن أيضاً ان تساعدك على ان تجد موضوعات محددة عند التصفح .

Tip



**على القرص المدمج:** قرب نهاية كل باب توجد ملخصات عن الموضوعات الموجودة فى الباب . وفى نهاية كل باب ، سوف تجد أيضاً مشروعات مقترحة وافكار عديدة للمستخدم الخبير . لمعرفة القصة الكاملة لـ component ما ، وابحث عنه فى المحتويات والفهرس . عليك أيضاً بالتعامل مع Online visual componeut المتواجد على القرص المدمج .



وتذكر : ان كل ال component classes مشتقة من classes أخرى ، لذا لكى تفهم كل الخصائص وال methods المتوفرة لـ component ما ، قد يتعين عليك القراءة عن classes عديدة .

## أى من نسخ Delphi يمكنك استخدامها؟:

أغلب الأمثلة فى هذا الكتاب، تستخدم Delphi Version 3 ونسخة متقدمة من Delphi Version 4، والتي تعمل فى الـ Windows 95. تنطبق بعض معلومات هذا الكتاب على كل نسخ Delphi ولكن، للحصول على افضل النتائج، يجب ان تقوم بتشغيل Delphi3 أو Delphi4. والعناصر الجديدة أو العناصر التى تعمل بطريقة مختلفة مشار إليها فى النص. اذا كنت تستخدم Delphi Version1 أو Delphi2، فإنك غالباً ستواجه مشاكل فى تشغيل العديد من التطبيقات يمكنك استخدام نسخة حديثة لكى تبدأ. إننى لا اشرح استخدام Delphi1 مع هذا الكتاب الا، ربما، للبداية فقط. يمكنك استخدام Delphi2 مع كثير من هذا الكتاب، ولكن احبذ لك التحديث سريعاً الى Delphi3 أو Delphi4 (تعمل اغلب معلومات والتطبيقات بهذا الكتاب مع Delphi3). اما المميزات الفريدة فى Delphi فقد تم الاشارة إليها.

## كيف تصل الى الكاتب؟:

بالطبع، انا لا اقع فى خطأ (انت تصدق هذا، اليس كذلك؟)، ولكن اذا اكتشفت خطأ فى هذا الكتاب، اكتب لى رسالة على عنوان الناشر أو ارسل لى خطاب بريد الكترونى الى [TomSwan@copuserve.com](mailto:TomSwan@copuserve.com). من فضلك لجعل رسالتك قصيرة. من فضلك ابعث بعنوانك.



# الجزء الأول

## مقدمة

محتويات هذا الجزء:

- الباب الأول: معلومات حول Delphi 4
- الباب الثاني: معلومات حول Visual Components
- الباب الثالث: معلومات حول Forms

كما ستعرف من خلال هذا الكتاب، وبمساعدة برنامج Delphi، يمكنك تطوير تطبيقات يصعب تطبيقها في وقت قصير بغض النظر عن خبرتك في مجال البرمجة.

إذا كنت قد بدأت الآن في تطوير برامج من خلال بيئة الـ Windows، فإنك قد اخترت النقطة الصحيحة لبدء منها. أما إذا كنت تعرف معلومات مسبقة عن الـ Delphi أو إذا كانت لديك الخبرة في كتابة البرامج لـ Delphi فيمكنك قراءة الأبواب الموجودة في الجزء الأول قراءة سريعة- ولكن لا تهمل قراءتها. فإن تلك الأبواب تشمل محتويات الكتاب وتحتوي على المعلومات التي تحتاج أن تعرفها قبل أن تتناول موضوعات أكثر تقدماً.

إذا كنت تستخدم إحدى النسخ الثلاث السابقة لـ Delphi- وخاصة إذا كنت تريد التعامل مع (Delphi)- أنظر جزء "لمحة تاريخية" في الباب الأول لتقارن بين الخصائص البرمجية للنسخ السابقة لـ Delphi لترى قائمة بالعناصر الجديدة من خلال (Delphi).



# الباب الأول

## معلومات حول Delphi 4

### محتويات الباب:

#### • لمحة تاريخية

#### • تطوير تطبيقات الـ Delphi

#### • البرمجة من خلال الـ Delphi Components

#### • وظيفة Code insight

#### • الـ Module Explorer

#### • الملفات وإمتدادات اسم الملف

#### • الـ Project Manager

إن البداية دائماً تكون أصعب جزء في عملية تعلم استخدام أداة تطوير برمجيات جديدة. ولكن، على الرغم من ذلك، يمكنك مع Delphi أن تنشئ تطبيقك الأول في دقائق. في هذا الباب أقدم لك الخطوط الرئيسية للكتاب، وركز على الخصائص الرئيسية للـ visual programming الخاصة بـ Delphi.

### لمحة تاريخية:

لقد مر برنامج Delphi حتى الآن بأربعة أجيال ولقد زادت كل نسخة جديدة من درجة التعقيد. وبسبب التغييرات الكثيرة التي أجريت لـ Delphi، فقد تحتاج بعض التطبيقات القديمة والـ Components السابقة الى مراجعة قبل أن تستطيع أن تتعامل معها من خلال Delphi.

على الرغم من ذلك ، فإن هذه التغييرات تعتبر بسيطة فى غالبية الحالات . فى الواقع ، لقد تمكنت من تطبيق مشروعات هذا الكتاب المتعددة مستخدماً Delphi 4 بلا أية تعديلات رغم أننى كتبته أساساً باستخدام نسخة Delphi الأولى . وللمق ، إن التطبيقات من خلال هذا الكتاب تعتبر بسيطة نسبياً ، إلا أنها تغطى غالبية خصائص البرمجة فى Delphi إلا أنه بالنسبة الى التطبيقات الخاصة بنظم قواعد البيانات أو OLE فإنه يفضل إعادة كتابتها مرة أخرى .

وأحد أسباب التغيير فى كتابة البرنامج والذى يعتبر سهلاً بالنسبة لبيئة Delphi ، هو أن الـ object class قد تم إضافة بعض الخصائص إليها ، ولكن لم يتم تغييرها بصورة رئيسية . ولقد تمت مراجعة الخصائص بالـ object ، وذلك بشكل أساسى فى غالبية الحالات ، ولكن خصائص class لم تتغير بشكل قوى لذلك ، فإن البرامج التى تستخدمها تحتاج فقط الى إعادة عملية compile لإضافة بعض الخصائص الجديدة للتطبيق الخاصة بـ Delphi 4 ، وتعد هذه إحدى المميزات الخاصة بالـ object-oriented programming .

### الخصائص الرئيسية فى Delphi 1:

إن النسخة الأولى من Delphi ، والتى تم إصدارها فى نفس الوقت مع Windows 95 ، قدمت ما يسمى بـ visual programming . وبالرغم من وجود الـ Visual Basic (VB) لفترة ، إلا إنها ما زالت تنتج code تحت سيطرة الـ interpreter ولكن Delphi تقوم بعملية الـ Compile للـ Source code .

**ملحوظة:** ينشئ Delphi 1 تطبيقات للنظم ذات ١٦ Bit ليتم تشغيلها فى برامج Windows 3.1 . وإنشاء تطبيقات للنظم ذات ٣٢ Bit لم تنفذ إلا فى Delphi 2 .

Note

والخصائص الرئيسية لـ Delphi 1 لا تزال موجودة دون تعديل وهذه الخصائص كما يلى :

• **إضافة visual components على الـ form:** تستطيع أن تضع VCL مباشرة على FORM ، وكل component هو عبارة عن object نستطيع أن نحدد معنى الـ component من ناحية البرمجة بأنه Object Pascal class .

## الباب الأول : معلومات حول Delphi 4

### ● Object Inspector Window - property editor : يستخدم لتحديد

المواصفات الخاصة بكل ال components مثل العنوان الخاص بال component ، وحجمها ، وكذلك لونها .

### ● event editor : ويتم اختياره أيضاً بواسطة ال Object Inspector - ويتم

إضافة الجمل المراد تنفيذها من خلال زر عند تنفيذ مجموعة من الجمل تختار من ال event editor ال event property المخصص لتنفيذ حدوث الضغط على الزر .

### ● Delphi : automatic code generation يكون class والتعريف

للمتغيرات و event-handler procedure فارغة عند بدء تكوين تطبيق جديد ، أضف ال objects على ال form وأختر ال object properties لتكوين ال event handlers والتي تملأ بال code الخاص بك .

### ● Native .exe code files : التطبيقات عند إجراء عملية compile

تحتوى على كافة VCL التي تنتمى الى Project ، مما يزيد حجم الملف ذا الإمتداد exe وأيضاً ، يجعلها فى بعض الأحيان مضيفة لمساحة القرص ، ولكنها سريعة للغاية ، وأصغر بكثير من التطبيقات الشبيهة التي تعمل بـ C++ class libraries والتي تستطيع إنتاج ملفات ضخمة ذا الإمتداد exe ، (إن Delphi تحتوى على smart linker ولغة برمجة جيدة يؤدى الى إنتاج تطبيقات جيدة) وعلى النقيض من ال VB ، أنه لا توجد حاجة الى إنتاج run-time interpreter عند استخدام أى تطبيقات كتبت بـ Delphi .

### ● إختفاء ال pointer : لم تختفى ال Pointers تماماً فى الواقع ، فهي ما زالت

تستخدم وبشكل قوى فى ال Object Pascal . ولكنه لم يعد ضرورياً استخدام pointers بشكل كبير فى معظم الحالات . السبب الرئيسى لهذا هو أنه مع وجود بيئة VCL ، وأصبح التحكم فى الذاكرة مسئولية ال component libraries بدلاً من واضع البرامج . لذا ، فنادرأ أن يكون من الضروري تخصيص ذاكرة Objects واستخدام pointers للتعامل معها . فإن Objects يتم إنشائها عند الحاجة ، وهذا يضع ال pointers فى عالم النسيان ولكن ما زالت موجودة عند الحاجة الى استخدامها .



• **إمكانية استخدام الـ Visual Basic (VBX):** لا يمكنك إنشاء الـ VBX من خلال Delphi 1، ولكن يمكنك استخدام VBX، وذلك من خلال تحويلها إلى component تستطيع التعامل معه.

• **إنشاء نظم قواعد البيانات باستخدام الـ visual components والـ Borland Database:** إن إمكانية كتابة تطبيقات متميزة خاصة لإنشاء كافة عناصر نظم قواعد البيانات تعد العنصر الرئيسي في نجاح Delphi الساحق.

هناك نسختان متوفرتان من Delphi 1-، هما الـ desktop والـ client-server. ونسخة الـ desktop مصممة لتطوير تطبيق standalone Windows التي لا تتعامل مع الشبكات. ولكن نسخة client-server، تعطيك إمكانية التعامل مع الشبكات، هذا بالإضافة إلى الوصول لـ mainframe database من خلال تطبيقات Delphi. ولقد أصبح Delphi الآن متوفر بثلاث نسخ الـ desktop، professional، والـ client-server.

إن الانتقال من البرمجة باستخدام الـ Turbo Pascal و Borland Pascal إلى Delphi 1 يتطلب إعادة النظر في عملية الانتقال. حيث، أن غالبية التطبيقات يجب إعادة كتابتها من البداية، مع احتمال استخدام أجزاء من التطبيقات من خلال Delphi، هذا بالإضافة إلى استخدام Delphi لمفهوم visual environment واستخدامه لـ VCL. فإن (VCL) Visual Component Library التابعة لـ Delphi 1 ليست متوافقة مع الـ Borland Pascal Object Windows والـ Turbo Vision السابقة لها. ولهذا السبب، فإن استخدام تطبيق كتب بهما من خلال Delphi 1 هو أمر غاية في الصعوبة.

### علاقة Delphi بـ OWL

من الناحية الفنية، كان Delphi 1.0 متوافقة بنسبة ١٠٠٪ مع OWL و Turbo Pascal for Windows (TPW). في الواقع كان OWL تصدر مع Delphi 1.0 حتى يتمكن المطورين من إجراء عملية compile لبرامجهم بلا تعديلات. ولكن لتحويل تطبيقات OWL إلى تطبيق يعمل مع Delphi التي تستخدم الـ VCL الجديدة كان هذا يعني إعادة كتابة غالبية التطبيقات من البداية، الأمر الذي دفع معظم المطورين إلى عمله.

## الباب الأول : معلومات حول Delphi 4

هناك سببان لإفصال Borland عن طراز الـ OWL. السبب الأول هو أنه لم يكن ملائماً تماماً لفهرم (VCL) ولأنه يفقد إمكانية التعامل مع properties و methods- وهي مكونات هامة للـ Components. أضف الى ذلك أن OWL كان يحتاج الى تطوير كبير لينتقل الى نظام الـ ٣٢ بت، لذا اختار Borland الخيار المنطقي، وإن كان مؤلماً، في أن يفصل عن الـ OWL وأن يقدم (VCL) مع خصائص الـ Object Pascal لتدعيمها. (في بعض أبواب هذا الكتاب، وعلى القرص المدمج، سوف نجد أيقونة الـ OWL الأصلية الخاصة بي والتي كانت معي منذ الـ (TPW 1.0).

وفي ملاحظة أخيرة، إن تطبيقات DOS التي كتبت مع الـ Turbo Pascal قد تم تحديثها بسهولة الى تطبيقات الـ Win32 باستخدام Delphi 4.

إذا كنت تستخدم Delphi 1، فسوف نجد بعض المعلومات في هذا الكتاب مفيدة لك. ولكن، عليك إجراء تغييرات لكثير من البرامج الموجودة، وبالطبع، الخصائص الموجودة في النسخ التالية غير متاحة لك. ولا اقترح عليك استخدام Delphi 1 مع هذا الكتاب إلا لمجرد البدء فقط.

### الخصائص الرئيسية في Delphi 2:

إن الميزة الرئيسية في Delphi Version 2 هي تدعيم إمكانية البرمجة بنظام ٣٢ بت. والخصائص الجديدة الأخرى في Delphi 2 تتضمن:

● **Long and short strings:** أخيراً تم التخلص من مشكلة الـ Borland Pascal- وهي الـ string ذو ٢٥٥ حرف طويلاً والذي يطلق عليها "length-byte" وهذا النوع للـ string ما زال موجوداً، ولكن long strings حررت واضعوا البرامج أخيراً من الإضطرار الى عدد لا حصر له من الأساليب الكثيرة لتخزين long strings في الذاكرة.

● **32-bit integer:** والذي جعل لا بد من إعادة النظر الى المتغيرات التي يكون الـ ١٦ بت طويلاً لها.

● **All cod and data references are now far (32 bits):**

وهذا لا يؤثر على برمجة التطبيقات إلا في الحالات التي تكون فيها مستخدماً للغة assembly- وهذه تكون خطيرة على البرنامج.

• الأماكن التى يتم حجزها فى الذاكرة محدودة بكم الذاكرة المتاحة فقط والذي قد يكون ذا حجم كبير من ال megabytes.

• Huge arrays: يصل حجم تكوينها ما يوازي حجم 32 bit index.

• استخدام ما يسمى بال multithreaded.

• التعامل مع مكونات ال Windows ذات ال ٣٢ بت مثل TreeView و outlines.

• إلغاء إمكانية التعامل مع ال VBX ذات ال ١٦ بت، وإضافة إمكانية التعامل ال OCX ذات ال ٣٢ بت.

• تحسين إمكانية التعامل مع ال Object Linking and Embedding (OLE).

• تحسين إمكانية التعامل بين Borland Database Engine و SQL.

• صدرت عدة نسخ مثل ال developer، desktop و ال client-server.

• التعامل فقط مع Windows 95 أو NT. حيث لم تعد التطبيقات الخاصة بهذه النسخة تعمل مع Windows 3.1.

من خلال مسار التطور، من Delphi 1 الى Delphi 2 لن تجد صعوبة لأن ال classes الموجودة لم تتغير الى حد كبير. وإن السبيل الى تحديث تطبيق ما هو إلا أن تحول تفكيرك الى نظام ال ٣٢ بت ويتم تحديث أى برنامج من حيث pointers و 16-bit integer، ويجب تحديث أية procedures أو functions الى ال ٣٢ بت. على الرغم من ذلك، هذه الخصائص الجديدة تعتبر أكثر آلية لغالبية واضعى البرامج الذين يستخدمون Delphi ل visual environment لإنشاء التطبيقات.

فى غالبية الحالات، يعتبر إعادة عملية ال compiling للبرنامج وكذلك components هو كل ما يحتاج إليه للإرتقاء من Delphi 1 الى Delphi 2.

ولا ينطبق هذا على تطبيقات Delphi 1 التى تستخدم ال VBX ذات ال ١٦ بت، والذي قامت Borland بتقديمها كعينات غير متوفرة فى Version 2. وهذا

## الباب الأول : معلومات حول Delphi 4

يتضمن البرامج التي تستخدم مكونات الـ TBiGauge ، TBiSwitch ، TChart ، و TBiPict .

**تحذير:** إن تجربتنا مع الـ VBX ذات الـ ١٦ بت تشير إلى أنه ليس من الحكمة استخدام Delphi Sample components في تطوير التطبيق . إنني هنا أشير إلى الـ component الموجودة في VCL palette Samples وكل الأمثلة من البرامج والتي تأتي مع Delphi ، ننوي استخدامها في البرنامج النهائي .

إذا كان تطبيقك يستخدم أية resources ذات النظام الـ ١٦ بت ، فسوف تحتاج أن تعيد عملية الـ compile لها وباستخدام linker (32-bit) لتكون الملف ذي الإمتداد (exe) النهائي . على سبيل المثال ، إذا قمت بإدخال cursor bitmaps ، فعليك استخدام الـ Image Editor لإنشاء resources ذي ٣٢ بت جديد وذلك لاحتواء التطبيق للـ cursor images .

يمكنك استخدام Delphi 1 مع هذا الكتاب ، ولكن اقترح عليك الإرتقاء بأسرع ما يمكن إلى Version 3 أو Version 4 .

### الخصائص الرئيسية في Delphi 3:

تستطيع أن تلقب هذه النسخة بأنها النسخة التي عززت مركز Delphi كأعلى نظام تطوير في الـ Windows . والخصائص الجديدة في Delphi 3 تتضمن:

• **code packages:** والذي مكن التطبيقات المتعددة من التشارك في كتابة البرنامج . والفائدة الرئيسية هي أنه لم يعد ضرورياً لكل تطبيق تمت له عملية الـ compile أن يحتوي على الـ VCL ، كما كان الحال في النسختين السابقتين . الـ code packages تعني أن مطوري الـ components يجب أن يراجعوا منتجاتهم ويعدوهم للاستخدام من قبل المستخدم- وأصبحت عملية تطوير بناء component كتطوير التطبيقات تماماً .

• **Numerous enhancements to the VCL:** وتشمل Win32

components الجديدة . وكذلك تتوفر الـ charting و third-party .

• **تدعيم كامل لـ ActiveX، بما فى ذلك إنشاءها،** ويضيف Delphi 3 إمكانية وجود لـ native language مما يسهل استخدام وإنشاء الـ COM Objects من خلال Delphi.

• **القدرة على أداء عملية compile للـ units وتحويلها الى الملفات ذات الإمتداد (.obj):** هو تغيير طال انتظاره منذ عهد الـ Turbo Pascal. وهذا يسمح للـ compiled module بأن يتم إدخالها فى نظم تطوير أخرى مثل الـ C++.

• **multitiered database و client-server component** تم تدعيمهما من خلال الـ TClientDataset والـ TRemoteServer الجديدة؛ وهذه الـ component تتوفر فى طبعة الـ client-server فى Delphi3 فقط.

• **عناصر الـ Internet و intranet** تم تكوينها من خلال نسخة الـ professional و الـ client-server؛ وتتصل هذه الـ component بالـ server الخاص بشبكة Web وتقوم بتبسيط تطوير تطبيق الـ Web وذلك بأن تخلص واضعى البرامج من ضرورة الإلمام الـ HTML communication protocols.

• **بنية تطوير متكاملة ويطلق عليها (IDE) والـ code insight؛** وهذا يضيف خصائص كانت فى الماضى متاحة لمحررى البرمجة الثانويين فقط. ولكن شكل Delphi على الرغم من ذلك لم يتغير كثيراً.

• **تدعيم إمكانية WideString character-string type؛**

• **الربط بين الـ Borland Database Engine و SQL التى تم مراجعتها حديثاً للحصول على تطوير قاعدة بيانات متطورة؛**

• **إصدارات الـ Delphi مثل الـ Standard أو الـ professional والـ client-server؛** ان غالبية الـ components وادوات تطوير قاعدة البيانات تتوفر فى جميع الإصدارات. ولكن طبعة الـ client-server توفر ادوات لإنشاء تطبيقات database multities. باستخدام الـ remote SQL servers والشبكات

## الباب الأول : معلومات حول Delphi 4

المحلية . وتوفر نسخة الـ professional امكانيات إنشاء تطبيقات على الشبكات المحلية وكذلك توفر الدعم للتطبيقات التي تستخدم ODBC بواسطة third-party drivers .

**ملحوظة:** في الواقع ، ان عالم Delphi 3 له بعد رابع . اذا كان لديك شغف للمعرفة فيمكنك الحصول على طبعة الـ Delphi Enterprise ، والتي تتضمن مزيد من الادوات لتوصيل تطبيقات Delphi 3 بمصادر قاعدة البيانات الى "data sources" باستخدام تكنولوجيا الـ enterprise الخاصة بـ Borland . أما السعر ؟ فعليك ان تسأل . . . . .

Note

لتحديث تطبيقات Delphi 2 الى Version 3 ، فيمكنك كالعادة إعادة عملية الـ compile دون صنع أية تغييرات . ولكن ، قد تريد الإنتفاع بمزايا الخصائص الجديدة مثل إمكانية الـ Internet وربما ، الـ ActiveX .

وبعض الـ Delphi 3 components موضوعه في أقسام مختلفة تحتويها VCL palette . وبعض الـ components الأخرى قد تم استبدالها بـ components مشابهة ، وإن كانت غير مطابقة . على سبيل المثال ، إن Header component لم يعد متاحاً على Additional component ، بل تم استبداله بـ HeaderControl ٣٢ بت ولكن يفتقد الى بعض خصائص الـ component السابق .

يمكنك استخدام Delphi 3 مع غالبية المعلومات والأمثلة الموجودة في هذا الكتاب . والخصائص الموجودة في Delphi 4 فقط دون غيره مميزة بوضوح أو مشار إليها في النص .

### الخصائص الرئيسية لـ Delphi 4:

إن مستخدمى أى من نسخ Delphi السابقة يجب أن يكونوا قادرين على الإرتقاء مباشرة الى Delphi 4 . والخصائص الجديدة في Delphi 4 تشمل البنود التالية ، ومعظمها موضح تفصيلاً من خلال هذا الكتاب :

• **شكل جديد ، ولكنه مألوف:** إن Delphi speed Bar و VCL palette ، واللذان كانا فيما مضى واقعين الى جوار بعضهما البعض ، أصبحتا الآن فوق

بعضهما . ولقد أصبح هذا ضرورياً لأن عدد component والأقسام قد زاد عن حجم الـ palette الأساسى بشكل كبير . مع Delphi 4 ، لم يعد من الضروري تحريك اللوحة يمينا ويساراً لتجد الـ component التى تحتاجها . (قد يظل هذا ضرورياً مع low resolution).

• **امتدادات لغة Object Pascal**؛ وهذا يؤدي الى عدة تغييرات فى الـ Object Pascal ، ولكن جميع التغييرات ذات طبيعة متطورة ومن غير المحتمل أن تقوم بإيقاف تنفيذ البرنامج الموجودة . وبعض التغييرات تجعل الـ Object Pascal أكثر شيوعاً بالـ C++ وأكثر قيمة لمطوري component الذين يريدون استخدام نسخ Delphi و C++ Builder لمنتجاتهم . وخصائص اللغة الجديدة تتضمن dynamic arrays ، method overloading ، وقيم ثابتة ، والنوع الجديد 46-bit Integer ، فى النوع Real ، والتغير فى بعض الأساليب الفرعية لإستدعاء Windows API وإستدعاء الثوابت أيضاً . انظر الباب الحادى والعشرين ، تطوير مهارات Delphi الخاصة بك ، للحصول على معلومات حول خصائص الـ Object Pascal .

• **Project Manager**؛ ان مدير المشروع القديم قد تم إعادة إنشائه كاملاً فى صورة tree-outline model ، مما يمكنك من متابعة مسار ملفات المشروع بسهولة . ولقد اصبح من الممكن الآن ايضاً إنشاء مجموعات مشروع ، والتى بإمكانها ان تقوم بتصنيف مشروعات متعددة بطريقة متطورة ويتم تخزين مجموعة المشروع فى صورة text Make file ويمكن بواسطة هذا ملف إجراء عملية compile على المشروعات من نظام التشغيل الـ "DOS" ولزيد من المعلومات انظر فقرة "Project Manager" فى هذا الباب .

• **Module Explorer**؛ ان code editor التابع لـ Delphi يتضاعف فى الـ IQ مع هذه الإصدار الجديدة . والـ Module Explorer ، الذى يوجد عادة الى اليسار من نافذة code editor ولا يمكن نقلها الى نافذة منفصلة ، ببساطة يتواجد فى ملف pascal source code unit ويظهر الـ Module Explorer كافة التقارير فى unit file ، التى يمكنك استقدامها لتحديد مكان أجزاء متنوعة من البرمجة فى النص . على سبيل المثال ، ان الـ Module Explorer يمنحك طريقة

## الباب الأول : معلومات حول Delphi 4

بسيطة للانتقال من تعريف method الى تنفيذ ذلك الـ method . لمزيد من المعلومات انظر فقرة "Module Explorer" فى هذا الباب .

• **Dockable Tool Windows** : ان نوافذ ادوات Delphi تعتبر Dockable . على سبيل المثال ، يمكنك Undock toolBar أو نافذة اخرى ، مثل الـ Module Explorer ، وتعيد عرضها كنافذة مستقلة عن الحاسب . يمكنك ايضاً ان تصل نوافذ لإنشاء نوافذ متعددة الصفحات . على سبيل المثال ، بدلاً من عرض نوافذ الـ Breakpoints والـ Watches منفصلة ، يمكنك توصيلها فى نافذة واحدة ثم اختر الادوات الفردية بالضغط على ابواب الصفحات ببساطة قم بضغط وسحب النافذة التى تريد توصيلها أو فصلها . ملحوظة : لتوصيل نافذة ، اضغطها واسحبها الى حافة نافذة أخرى ، مثل الـ code editor واستمر فى السحب حتى يتغير حجم الخطوط الخارجية للنافذة ، وفى أى نقطة يمكنك ترك الفأرة .

• **Improved Debugging** : هناك ثلاثة عناصر إضافية لعملية debugging .

نافذة CPU تظهر الـ low-level state لتطبيق ما ، بما فى ذلك قيم الـ register و flag الـ CPU . يمكنك ايضاً مراجعة وإزالة اخطاء تطبيقاتك على مستوى لغة اذا كانت assembly . وتظهر نافذة الـ Module الآن الـ module الخاصة بالتطبيق ، قائمة بملفات التى تمت عملية الـ compile لها لإنشاء module وقائمة الـ global symbols و entry points وتقدم النافذة الثالثة الجديدة نافذة الـ logEvent التى توضح رسائل process control ورسائل الـ breakpoints ، رسائل الـ Output-Debug-String ، ورسائل النافذة . ويحتوى Delphi 4 ايضاً خصائص إزالة اخطاء متقدمة مثل الـ multiprocessing ، remote debugging ، debugging of CORBA ، cross-language debugging ، debugging .

• **زيادة عدد VCL** : لقد إزدادت عدد components VCL فى Delphi 4 . وهى تشمل قوائم أوامر لربط الاستجابات الخاصة بالـ objects من الناحية الوظيفية مثل القوائم والازرار ، امكانية وجود بعد التطبيقات خاصة بالـ Windows NT ، وعناصر الـ TControl والـ TWinControl لإنشاء نوافذ و toolbars والتحكم عند حدوث تغيير فى حجم النوافذ أو عند حدوث تغييرات أخرى وتتضمن نسخة



ال client-server ال TQueryClientDataSet component الجديدة  
و component أخرى .

• **برمجة Multitiered Database**، تزيد ال components الجديدة من  
قدرة برمجة بيانات ال client-server . فقد تم تحسين عملية الاتصال من خلال  
Socket server mconnect unit والتي تدعم ال call backs و socket server  
application يتم تنفيذها ك server . Windows NT . وتشمل خصائص قاعدة  
البيانات المتقدمة الأخرى إمكانية وجود ال Oracle8 Abstract Data Types  
(ADTs) كحقول لل Table .

• **دعم component (MTS) Microsoft Transaction server**،  
ان database classes فى Delphi 4 ، و COM class ، وحتى بيئة التطوير  
المتكامل لـ Delphi تساعد على إنشاء ، تركيب ، و debugging لل com objects  
وتحت سيطرة MTS . وهناك أيضاً إمكانية إنشاء ، إزالة أخطاء والتعامل مع  
CORBA objects .

• **Database Programming Enhancements**، يمكنك تصفح  
قواعد البيانات وتعديل ال Tables بشكل ايسر باستخدام TTable dataset  
component . اضغط زر الفأرة اليمين لـ TTable object لتتمكن من الوصول  
الى field editor ولكى تحذف ، وتعيد تسمية ، وتقوم بتحديث هياكل ال Table .  
يمكنك أيضاً تشغيل ال Database Explorer من هذه القائمة . اذا كان ال Table  
متصلاً بمصدر بيانات ، فهذا يعرض الجدول ومعلوماته فى ال Explorer .

• **ما تم إضافته الى ActiveX**، لقد قامت Delphi بزيادة كفاءة ال  
ActiveX لتشمل لـ Visual Basic data interfaces وأيضاً توفير إمكانية إنشاء  
nonvisual ActiveX .

اذا كان لديك Delphi4 ، فيمكنك استخدام جميع المعلومات وتشغيل كل  
الأمثلة الموجودة فى هذا الكتاب . يمكنك استخدام أى من إصدارات Delphi (أو  
professional) أو (desktop) ، فأنت لا تحتاج الى ال client-server ، رغم أننى  
قد ذكرت بعض قدراتها . (لقد دارت شائعة أن Inprise سوف يغير أنواع

## الباب الأول : معلومات حول Delphi 4

الإصدارات المطروحة مع Delphi4 ولكن هذا لن يؤثر على المعلومات التي ستحصل عليها من هذا الكتاب).

**ملحوظة:** إن المعلومات الموجودة في هذا الكتاب حول الخصائص الجديدة في Delphi 4 يمكن أن تساعدك في تحديد إذا ما كنت سوف تقوم بالتحديث من نسخة سابقة ، هذا إذا لم تكن قد اتخذت هذه الخطوة بالفعل أو تخشى المخاطرة بها. إذا لم يكن لديك Delphi 4 حتى الآن ، فيمكنك استخدام في Delphi 3 مع غالبية الأمثلة الموجودة في هذا الكتاب.

### تطبيقات Delphi المطورة:

كما هو الحال مع معظم برامج الحاسب ، إن أفضل طريقة للتعرف على Delphi هو استخدامه . ففي هذه الفقرة ، سوف تقوم بتجربة visual environment الخاصة بـ Delphi . إبدأ الآن مع Delphi يوضح شكل (١-١) Delphi وهو يعمل في Windows 95 إذا لم تنشئ شاشتك الشكل ، اختر أمر ال File | New Application .

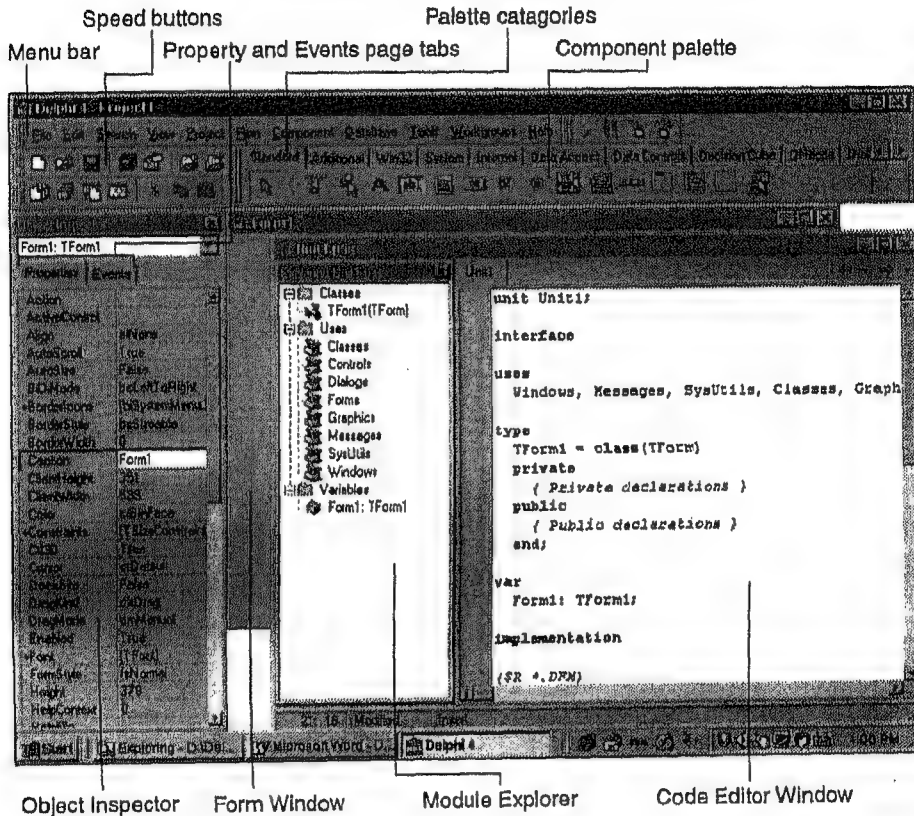
قد تختلف بعض الأزرار والنوافذ وهذا يعتمد على نسخة برامجك ، فنظام التشغيل ، وعناصر عملية installation . واستناداً الى نسخة الـ Windows الخاصة بك ، قد تختلف كثير من title bars ، borders ، buttons أشياء أخرى عن تلك الموضحة هنا . وتلك الفروق إنما هي فروق شكلية الى حد كبير ، ولا تؤثر على أى من المعلومات الواردة في هذا الكتاب إلا في المواضع التي أشرت إليها .

[Delphi [integrated development environment (IDE)] لها عدة عناصر أساسية . وتلك العناصر هي [حسب الترتيب الوارد في شكل (١-١)]:

● **Speed buttons:** هناك ما يسمى point-and-click buttons وهي مخصصة لإختيار الأوامر من القوائم . يمكنك أن تعرف ماذا يفعل الزر وذلك بوضع مؤشر الفأرة عليه (ولكن لا تضغط) وتنتظر لحظة لتري الـ hint box والذي يحتوى على اسم الزر . تمرن على استخدام الـ Speed Button الخاصة بـ Delphi والتي على المدى البعيد سوف توفر وقتك الذى تبذله فى عملية التطوير ، وخاصة اذا كنت تتحكم فى الفأرة بشكل جيد . على سبيل المثال ، بالرغم من أنه بإمكانك

## دلفى ٤ باييل

تشغيل برامج بطرق أخرى، إن أسرع وسيلة هي فى الغالب أت تضغط زر الـ "Run" (المثلث المشير الى اليمين). تم تغيير بعض الأيقونات فى Delphi 4 لتلائم بشكل أفضل مع توصيات الـ Microsoft على سبيل المثال، زر المشروع Save (الثانى من اليسار) يبدو الآن وكأنه مجموعة من الأقراص.



شكل (١-١) Delphi 4 وهو يعمل فى الـ Windows 95

• **Menu bar**: وهى قائمة تتشابه فى الشكل مع القوائم فى بيئة الـ Windows. قم بفحص قوائم Delphi الآن لكى تتعرف عليها. إن Delphi 4 يضيف أيقونات لكثير من الأوامر. إنها نفس الأيقونات المعروضة فى شكل Speed buttons فى الـ toolbar.

• **Component palette**: تحتوى هذه palette على أيقونات تمثل الـ component فى الـ VCL. اضغط على palette button لتختار visual

## الباب الأول : معلومات حول Delphi 4

component ثم اضغط الفأرة داخل ال form window لتقوم بإضافة component object للحصول على معلومات عن component معين، ضع مؤشر الفأرة على أى component button ولكن لا تضغط الفأرة. بعد لحظة، يقوم Delphi بعرض hint box يحتوى على اسم ال component قم بتحريك الفأرة لترى ال hint boxes عن أى component آخر. استخدم هذا الأسلوب لتحديد مكان component معينة متكررة فى هذا الكتاب.

● **Palette categories:** لرؤية المزيد من palette categories، اختر عنصراً من عناصر page tabs والذي يظهر بصورة واضحة أعلى VCL palette. على سبيل المثال، اختر عنصر Dialogs page لكى يظهر لك Delphi's dialog-box component.

● **Properties and Events page tabs:** اختر واحدا منهما وهما إلى الأعلى من Object Inspector window إما لتختبر properties window أو event window، ال properties window تمثل الخواص الخاصة لل component، مثل حجم الزر أو تحديد نوع الخط لل text label. و event window تمثل حدوث شئ ما، مثل الضغط لإحدى أزرار الفأرة أو ضغط أى زر من المفاتيح.

● **Object Inspector:** توضح هذه النافذة كل ال properties وال events لواحد أو أكثر من ال component أو forms المختارة. بمرور الوقت سوف تصبح صديقاً حميماً لهذه النافذة، فبالرغم من بساطتها الظاهرية، تعد نافذة ال Object Inspector واحدة من أهم أدوات البرمجة الخاصة بـ Delphi. (فكرة: اضغط F11 لإستدعاء نافذة ال Object Inspector اذا كانت مخبئة بنافذة أخرى، أو اذا كانت أغلقت فجأة).

● **Form:** فى كل التطبيقات تمثل ال form الأولى ال form الرئيسية للبرنامج ومايلى من نوافذ، تمثل نوافذ أخرى- على سبيل المثال dialog box أو a child window فى Multiple Document Interface (MDI). يمكن ان يكون هناك form وحيدة للبرامج البسيطة أما التطبيقات المعقدة فيكون لها عشرات ال forms وتساعدك ال dotted grid على تنظيم المسافات بين components

المضافة الى ال form . ولا تظهر ال dotted grid فى التطبيق النهائى . (فكرة: أضغط F12 مرة واحدة أو أكثر لتحديد موضع هذه النافذة اذا كانت تخفيها نافذة أخرى).

● **Module Explorer:** وهو الجديد فى Delphi 4 ، فهذه النافذة تظهر ال current module الحالية بالإضافة الى قائمة بال units الأخرى ، التى من الممكن التعامل معها من خلال إستدعائك ل variables أو objects أو methods ، وأى معلومات أخرى . للتوصل الى ال method فى ال code editor window ، فعليك ببساطة ان تضغط أى من العناصر المدخلة لل Module Explorer .

● **Code editor window:** تظهر هذه النافذة البرنامج المكتوب بلغة Pascal المرتبطة بكل form فى التطبيق . وسوف اطلق عليها code editor أو اسم editor window من الآن فصاعداً ويقوم Delphi بصورة تلقائية بإضافة التعريف وال events الى هذه النافذة ، التى يمكنك ان تضيف إليها جمل Pascal أو التى تنفذ مباشرة تبعاً لل events الذى يحدث مثل أوامر القائمة أو ال button clicks . يمكنك أيضاً استخدام هذه النافذة لتحرير Pascal modules أو عرض text files قم بالاختيار من بين الملفات المفتوحة بإختيار أحد عناصر ال page tabs المتواجدة أعلى ال editor window .

● **Windows 95 task bar :** يقوم ال Windows 95 task bar بعرض زر Start وايقونات التطبيقات التى يتم تحميلها حالياً . ولإعادة التعامل مع تلك البرامج المحملة ، عليك ان تضغط الزر الخاص به بتلك البرامج .

Tip **فكرة:** اضغط F 12 لتربط بين code editor ، form windows ، التى تختبئ وراء بعضها البعض .

سوف تعرف اكثر عن النوافذ ، الأوامر والامكانيات الاخرى فى Delphi كلما قمت بتجربة الأمثلة الموجودة فى هذا الكتاب . كما انك ستقابل الكثير من النوافذ الأخرى مثل ال Project Manager ، ال Object Browser ، ال Integrated Debugger ، ال Code Editor ، ال Code Insights ، ال

## الباب الأول : معلومات حول Delphi 4

Image Editor ، Package Editor ، Menu Designer ، وادوات أخرى . حتى تصبح أكثر تعرفاً على Delphi ، يجب ان تعيد ترتيب عرض شاشتك حتى تشبه الشكل (١-١) .

**فكرة:** قبل ان نستمر ، اختر Tools|Environment Options واختر Preferences page tab ، وأختر عنصر component captions هذا يعطى معلومات أكثر عن ال components رغم ان واضعى البرامج ذوى الخبرة يرفضون هذا الخيار . ان Delphi لا يظهر captions على جميع ال components ولكن فقط على أولئك الذين ليس لهم شكل متميز . على سبيل المثال ، مع اختيار هذا الخيار ، يتم تسمية DataSource component (الذى تستخدم فى برمجة قاعدة البيانات) ب DataSource1 أو باسم آخر اذا قمت بتغييره . وبدون caption على component فسوف تظهر component على انها ايقونات ، مما يجعل من الصعب التمييز بينهم .

Tip



### إعداد تطبيق جديدة:

ان الخطوة الأولى بعد البدء فى تطبيق جديد هى ان تعطيه اسماً . اننى احب ان اقوم بهذا عقب اختيار ال File|New Application مباشرة . ان تسمية مشروع جديد باسرع وقت ممكن يمنع Delphi من حفظ المشروع فى أدلته تحت اسماء ملفات افتراضية للبرنامج ، مما يهدر مساحة القرص .

ويتكون مشروع Delphi المثالى من عدة انواع من الملفات . تحتوى بعض الملفات على text ، واخرى تحتوى على binary values ، bitmaps ، codes . ولان كل تطبيق يتكون من عدة ملفات ، فإنه من الصواب إنشاء دليل منفصل على قرصك الصلب لكل مشروع جديد . بهذه الطريقة . يمكنك بسهولة نسخ ملفات تطبيق على قرص مرن أو محرك شبكة لحفظه بطريقة آمنة ، ويمكنك ان تحذف التطبيقات القديمة بإزالة الأدلة التابعة لها .

**فكرة:** ان حفظ المشروعات فى أدلة منفصلة يساعد على حفظ application modules بشكل منظم . لا تقم بتخزين مشروعات متعددة فى نفس الدليل - ان المجموعة الناتجة من الملفات سوف تصبح من الصعب فصلها عن بعضها .

Tip



لإنشاء أول تطبيقاتك فى Delphi، قم باداء الخطوات التالية الآن :

١ - استخدام الـ Windows Explorer لإنشاء دليل جديد (يمكنك ان تسميه مجلد) مثل C:\Projects لتخزين مشروعات Delphi الخاصة بك . يمكنك استخدام drive غير الـ C واسم آخر اذا كنت تفضل هذا . وفى داخل دليلك الجديد، قم بإنشاء دليل فرعى اسمه Hello . والمسار التام، C:\Projects\Hello جاهز الآن لحمل ملفات المشروع .

٢ - إرجع الى Delphi واختر FileNew Application . وبالتبادل، يمكنك اختيار FileNew، اضغط Page tab، واختر ايقونة الـ Application - ولكن هذه هى الطريقة الطويلة للوصول لنفس النهاية . اذا تم تحميل وتعديل مشروع آخر، فإن Delphi يعطيك فرصة لحفظ المشروع او للتخلص منه .

٣ - قبل إضافة component أو عمل أى تغييرات أخرى فى المشروع الجديد، اختر FileSave All (اضغط Alt+FV)، أو اضغط Save All button. إنك ترى two dialogs، واحداً تلو الآخر . الأول، والذي يحمل اسم Save Unit1 As، يطلب اسم ملف لملف source code الرئيسى الخاص بالـ Unit. واحب ان اطلق على هذا الملف اسم Main، وهذا يشير الى انه يحتوى على source code الخاص الـ form الرئيسى للقيام بهذا، افتح الدليل الذى انشأته فى الخطوة الأولى (Step 1)، واكتب Main داخل مربع ادخال اسم الـ File الخاص Save dialog. اضغط Enter أو اضغط Save لإنشاء ملف الـ Main.pas فى الدليل المفتوح . ويقوم Delphi تلقائياً بإلحاق امتداد اسم الملف pas. اذا لم توفر واحداً. ولا تقم بتغيير هذا الامتداد أو أى امتداد آخر بلا تمييز . ان Delphi يتعرف على انواع الملفات من امتداداتها، واذا قمت بتغييرها، فان Delphi قد لا يكون قادراً على انشاء التطبيق النهائى .

٤ - ثم يظهر Delphi بعد ذلك dialog آخر يحمل اسم Save Project1 As. وأود ان اطلق على مشروعاتى نفس اسماء أدلتها . على سبيل المثال، فى هذه الحالة، ادخل Hello لإسم ملف المشروع، ثم اضغط Enter أو اضغط Save. وهذا يخلق ملف المشروع الرئيسى، Hello.dpr، فى الدليل المفتوح يوفر Delphi تلقائياً امتداد اسم الملف dpr، والذي ترمز الى Delphi Project.

## الباب الأول : معلومات حول Delphi 4

**تحذير:** اذا لم يطلب Delphi اسم المشروع في الخطوة الرابعة، فقد يكون بطريق الخطأ قد اخترت أحد أمرى ال File|Save الأخرى، والذي قد لا يقوم بحفظ المشروع بأكمله. اذا حدث ذلك، فببساطة كرر الخطوات ٣ و ٤. في الخطوة ٣، يمكنك اختيار (Alt+FE) Save Project As...، لكننى أرى انه من الأيسر حفظ مشروعاتى وأى ملفات متعلقة باختيار File|Save All (Alt+FV).

لتطوير مزيداً من تطبيق Delphi الأول الخاص بك، استمر مع التعليمات الموجودة فى الفصل التالى.

**فكرة:** لضمان ان كل ملفات المشروع قد تم حفظها، اختر ال **Tip** Tools|Environment Options، اختر Preferences page tab، قم بتشغيل خيارى ال Autosave وهما ملفات Editorm وال Desktop. بهذه الطريقة، فإن أية تغييرات للملفات المشروع يتم حفظها تلقائياً فى كل مرة قبل ان تقوم بتشغيلها. لا تختبر هذه الخيارات رغم ذلك، اذا أردت عمل تغييرات مؤقتة وقمت بإدارتها لإختيار نظريات وافكار.

### تحديد ال window caption:

سوف نجد ان Delphi قد وضع مسمى لل caption وهو Form1 وهو خاص لل Form الرئيسية للمشروع Hello وتجد دائماً ال caption عبارة عن جزء من أى Form ويظهر دائماً فى الجزء العلوى لل Form. ولتغيير هذا المسماة تتبع الخطوات التالية:

١- يجب ان تعرض نافذة ال Object Inspector ال Form1: TForm1 فى أعلاها- اذا لم يكن الأمر كذلك، اضغط داخل نافذة Form1 (تلك التى تحتوى على dotted grid). اضغط F11 و F12 كما تحتاج لإستدعاء النوافذ اللازمة لتراها. فى أعلى نافذة ال Object Inspector [ارجع الى الشكل (١-١)]، اضغط Properties page tabs لعرض ال properties form، وهذا أيضاً يختار نافذة ال Object Inspector فتستطيع صنع تغييرات لل events ولل object properties.



٢- اختر خاصية ال Caption. (يمكن ان يكون قد تم اختيارها بالفعل حسب النظام الافتراضى للبرامج).

٣- لتعديل خاصية ما، اكتب أو اختر قيمة جديدة فى العمود الواقع الى اليمين من اسم الخاصية. على سبيل المثال، قم بتغيير قيمة ال Caption من Form1 الى Hello Delphi Programmer. فقط إبدأ الكتابة - وعليك ان تضغط Enter بعد الكتابة لاحظ ان caption يتغير وانت تكتب.

٤- احفظ المشروع باختيار FileSave All. ولأنك قد قمت بالفعل بإعطاء اسماء ملفات، فان Delphi ينفذ الأمر على الفور - أى انه لا يعرض save dialog التى رأيتها عندما قمت بإنشاء المشروع أول مرة. ولكن، فى كل مرة تضيف فيها Form جديدة للمشروع، فسوف ترى واحداً أو أكثر من هذه ال dialog مرة أخرى عندما تحفظ المشروع.

**فكرة:** عندما يتم حفظ المشروع تماماً، فإن Save All speed button سوف تصبح فى حالة dimmed. وهذه طريقة سريعة للتأكد بصرياً من ان أى تغييرات لل project modules قد تم تخزينها بشكل سليم على القرص.

لإنهاء تطبيقك الأول، استكمل التعليمات فى الفصل التالى.

### تشغيل التطبيق:

صدق أولاً تصدق، إنك قد انهيت من برمجة تطبيق Delphi الأول الخاص بك لتكملة عملية التطوير وتشغيل البرنامج، اتبع هذه الخطوات:

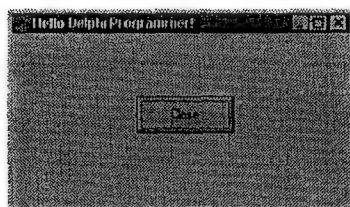
١- اضغط F9، أو اختر امر ال Run من قائمة ال Run أو، اضغط Run button (المثلث المواجه اليمين). استخدم احدى هاتين الوسيلتين لتشغيل تطبيقك مراراً خلال التطوير، ولاختيار البرمجة الجديدة. ليس عليك ان تنهى مشروعاً قبل ان تشغله.

٢- ان تروس حاسبك تعمل، يدور القرص، يخرج دخان من تجويفات محرك القرص، وعقب ثوان قليلة، تظهر نافذة التطبيق الاول الخاص بك على الشاشة. يوضح شكل (١-٢) عرض Hello فى ال Windows 95. قد يختلف

## الباب الأول : معلومات حول Delphi 4

مظهر البرنامج اعتماداً على مواصفات نظام التشغيل الذي حددته عند إنشاءك لنظام التشغيل .

٣- لإنهاء التطبيق ، اضغط مرتين زر قائمة النظام في الركن الأيسر العلوي في النافذة ، أو اضغط مرة واحدة الزر واختر Close من قائمة النظام . إذا كنت تفضل استخدام لوحة المفاتيح ، اضغط Alt+F4 لإغلاق التطبيق وعد الى Delphi . أو ، يمكنك ضغط زر إغلاق النافذة في الركن الأيمن العلوي .



شكل (١-٢) ، عرضت تطبيق Hello

رغم أن تطبيق Hello لا يفيد كثيراً ، إلا أنه يعرض جانب مهم من برمجة Delphi . إن المشروعات تبدأ العمل كتطبيقات تامة . قارن هذا بنظام الـ nonvisual ، مثل تركيب الـ C++ التقليدي الذي قد يتطلب منك أن تكتب عشرات ، أو أكثر ، من السطور قبل أن تتمكن من البدء في اختيار برنامجك . إن برامج Delphi مستعدة للتشغيل في غالبية المراحل الخاصة بتطويرها .

**تحذير:** بعد تشغيل التطبيقات من داخل Delphi ، أغلقهم دائماً لترجع الى غط البرمجة . إذا تركت التطبيق مفتوحاً بالمصادفة ، فلن تكون قادراً على إختيار كثيراً من أوامر Delphi . إذا لم تستطيع التعامل مع Delphi - وخاصة إذا لم تستطيع رؤية نافذة الـ Object Inspector بضغط الـ F11 - فإن السبب من المحتمل ان يكون هو انك نسيت وتركت التطبيق الذي يعمل بدون إغلاق .



ابدأ تجربة أخرى سريعة : اضغط F9 لتعود الى Hello . لاحظ انه هذه المرة ، تظهر نافذة التطبيق اسرع من أول مرة قمت فيها بتشغيل البرنامج . يدرك Delphi أنك لم تقم باية تغييرات في المشروع ، لذا يقوم ببساطة بإعادة تحميل ملف الـ application code مباشرة . اغلق نافذة برنامج Hello الآن .

يمكنك أيضاً استخدام الـ Windows Explorer لتشغيل تطبيق Delphi. على سبيل المثال، افتح الـ Explorer، وتصفح دليل الـ C:\Projects\Hello، واضغط مرتين Hello.exe، أو قم بابرار اسم الملف واضغط Enter. ان مستخدمى برنامجك يستطيعون إنهاء التعامل مع الـ Delphi كما يفعلون مع برامج الـ Windows الأخرى. تذكر اغلاق البرنامج قبل الاستمرار.

من الممكن ان تستخدم الـ Windows Explorer لتشغيل أكثر من نتيجة للبرنامج على سبيل المثال، يمكنك تشغيل نسختين أو أكثر من برنامج Helleo. ويمكنك، رغم ذلك، ان تقوم بتشغيل البرنامج مرة واحدة فقط من داخل Delphi. لتشغيل البرنامج عدة مرات، يجب ان تستخدم امر الـ Run button. أو الـ Explorer. اغلق كل نسخ البرنامج Hello قبل الاستمرار.

ان تطبيقات التى تم إنتاجها بواسطة Delphi تتمتع بصفات بيئة الـ Windows كاملة - فانك تقوم بتشغيله بنفس الطريقة التى تديرها التطبيقات الأخرى. يمكنك ان تضغط وتسحب اسم ملف خاص بالتطبيق desktop، أو ان تنشئ shortcut للملف (اضغط واسحب مستخدماً زر الفأرة الأيمن)، ثم اضغط مرتين الأيقونة الناتجة. وأولئك الذين يحبون ان يؤدوا الأشياء بالطريقة الصعبة يمكنهم. كتابة اسم المسار الكامل للبرنامج C:\Projects\Hello\Hello.exe باستخدام امر الـ Run فى زر Start. يمكنك أيضاً ان تفتح نافذة DOS، وان تنتقل الى دليل المشروع، وان تدخل Hello لتشغيل البرنامج. الفكرة هى ان التطبيقات التى انشأها Delphi تعتبر برامج له نفس التشابه فى التنفيذ كما يحدث فى بيئة الـ Windows حيث أنك تقوم بتشغيلها بنفس الطريقة التى تشغل بها برامج الـ Windows الأخرى.


### عملتى الـ (Compiling) والـ (linking) الخاصتان للبرنامج:

عندما تقوم بتشغيل تطبيق بضغطة الـ F9، يقوم Delphi بعملية ثم عملية الـ compiles والـ link للبرنامج لإنشاء executable code file. ويحدث شيئان أساسيان الأول هو ان compiler الخاص بـ Delphi يترجم نص البرامج الى binary code. ثم يقوم الـ linker بضم binary code الى الـ modules الأخرى المطلوبة لأعمال البدء المتنوعة ومهام أخرى. ونتيجة ما حدث لمشروع Delphi ينتج

## الباب الأول : معلومات حول Delphi 4

ملف من نوع executable ويحمل نفس اسم المشروع ، ولكن ذو اسم ملف ينتهى بالإمتداد exe .

لإجراء عمليتي ال compiling وال linking لتطبيق ما، ولكن دون تشغيله، اضغط Ctrl+F9 أو اختر أمر ال Project|Syntax Check . يمكنك استخدام هذه الطرق للتأكد من أن البرنامج خالى من الأخطاء مثل أخطاء الكتابة البسيطة .

**فكرة:** استخدم ال Tools|Environment وقم بإختيار Show compiler progress . وهذا يعرض dialog يوضح التفاصيل المتنوعة لعمليتي ال compiling وال linking وهو مفيد خاصة لتحديد الأخطاء فى التطبيقات الكبيرة . 

وعلى عكس النظم الأخرى التى تتمتع بحزم ال visual programming ينتج Delphi (true native code)، وهذا يعنى أن البرامج التامة لا تتطلب مترجم وقت التشغيل ففى النسخ السابقة من Delphi، كان الملف exe. الناتج تام بنسبة ١٠٠٪، وكان هو الملف الوحيد المطلوب من قبل مستخدمى برنامجك . وهذا ما زال ينطبق على Delphi 4 . ولكن، مع بداية Delphi Version 3، تم ترتيب كثير من Delphi's Visual Component Library (VCL) فى package يمكن لتطبيقات أو أكثر التشارك فيها . وإعتماداً على أنواع التطبيقات التى تقوم بإنشاءها، فقد تحتاج الى run-time packages وقت التصميم أو وقت التشغيل (أو كلاهما) للمستخدمين النهائيين . إنك قطعاً سوف تحتاج الى فعل هذا، على سبيل المثال، اذا قمت بتطوير custom components لواقعى برامج Delphi آخرين .

وسبب آخر لتوزيع package هو لتقليل كمية ال code فى ملفات ال exe. ففى نسخ Delphi Versions 1 and 2، كانت كل ما يحتويه المشروع من code، بما فى ذلك ال code الخاص بـ VCL بأكملها، قد تم تجميعها فى كل ملف exe . اذا كان لديك ستة تطبيقات Delphi مختلفة، فإن ملفات ال exe. الخاصة بهم تكون خزنت ست نسخ منفصلة من نفس مكونات ال VCL . ومن الواضح أن إضاعة المساحة هذه أمر غير مرغوب فيه، رغم أن الملفات الناتجة تكون بسيطة وسريعة التشغيل . واليوم، يمكنك إختيار أن توزع ال code العامة مستخدماً

package run-time (كما هو الحال Visual Basic و Java) استخدم interpreters. ويوضح الباب العشرون، إنشاء Components مخصصة عن كيفية إنشاء ال package الخاصة بك.

**ملحوظة:** إن Packages تعد أشياء إختيارية على مستوى التطبيق في وقت التشغيل. اذا كنت تريد تجميع كل ال codes المستخدمة من قبل تطبيقك، اختر Project\Options...، اضغط Packages page tabs، ولا تختار الإختيار المسمى Build with runtime packages.

Note

## برمجة ال Component:

قبل الإستمرار في هذا الباب، دعنا نتوقف قليلاً لتذكر سريعاً هذه الأشياء. إنك تعرف الآن ثلاث تقنيات هامة لـ Delphi:

\* كيفية إنشاء وحفظ مشروع جديد.

\* كيفية تعديل خاصية مثل ال Caption.

\* كيفية عمل compile، وعمل link، وتشغيل تطبيق بضغط F9، أو بضغط الزر Run (المثلث المشير الى اليمين).

إنك سوف تستخدم هذه التقنيات في كل تطبيق تقوم بكتابته. وبالطبع، إن البرامج التي لا تفعل شيئاً سوى عرض عنوان لن تأخذ شعار الأكثر مبيعاً. ولإعطاء تطبيق Hello شيئاً ليفعله، يمكنك إدخال visual component، في نافذة البرنامج، وقم بإختبار قمة ال code، كما ستوضح الفقرات التالية.

**فكرة:** يقوم Open-file speed Button بفتح الملفات الفردية. ولتحميل تطبيق Delphi كامل، اضغط دائماً speed Button Open-project، أو اختر ال FileOpen... لفتح ملف مشروع ينتهي بالإمتداد .dpr. يمكنك أيضاً إختيار أمر ال FileOpen Project...، والذي ظهر ثانية في Delphi 4 بعد إختفاءه من Version 3.

Tip

## إضافة visual components:

اتبع هذه الخطوات لإضافة visual components فى نافذة برنامج Hello :

١- أغلق برنامج Hello العامل اذا كان ضرورياً للرجوع الى Delphi. اختر File|Open... أو ضغط زر Open-project. (قم بهذا لإكتساب خبرة حتى اذا كان Hello مفتوحاً بالفعل). عندما يظهر Open Project dialog، قم بتغيير الأدلة اذا لزم الأمر واختر Hello.dpr. اضغط Enter أو اضغط OK لفتح المشروع استخدم نفس الطريقة لفتح المشاريع الأخرى المذكورة فى هذا الكتاب.

**Tip** **فكرة:** اذا كنت تحتاج الى التوقف قليلاً أثناء قراءة الباب، احفظ المشروع ثم قم بإعادة تحميله عندما تستعد للإستكمال.

٢- اختر Standard page tab من components palette ثم اختر Button component الذى يحمل علامة Ok.

٣- حرك الفأرة داخل ال form واضغط مرة لإضافة ال Button object ولا يهم الموقع.

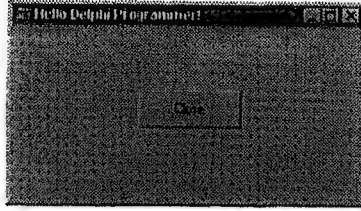
٤- تأكد من أنه تم إختيار ال Button object- يجب أن ترى أشكال مربعة تحيط به. اذا لم يتم إختيار ال object، اضغطه مرة الآن. وبعد ذلك يقوم Delphi بتسمية ال object بـ Button1. وللحصول على اسم يحمل معنى أكثر من هذا، اختر خاصية ال Name فى ال Object Inspector، واكتب CloseButton. إنها كلمة واحدة وليست إثنان- لا تضع أية مسافات بينها. اضغط Enter.

٥- لاحظ أن خاصية ال Caption ونص الزر يتغيران أيضاً ليصبحا CloseButton. وهذا أمر طبيعى، ولكنك سوف تحتاج دائماً الى التفرقة بين ال component object's name و caption properties، على سبيل المثال، اختر خاصية ال Caption واكتب Close. ويجب أن تشبه ال form الخاصة بك شكل (٣-١).

٦- اضغط مرة واحدة CloseButton object اذا لزم الأمر لتختاره. وفى أعلى نافذة ال Object Inspector، اختر Events page tab، والى تضع قائمة بالأعمال التى يمكن للزر القيام بها. اضغط مرتين المساحة الخالية الى اليمين من

حدث الـ `OnClick` . يظهر الـ `Main.pas` فى الـ `code editor` ، ويضع `Delphi` المؤشر الوامض بين الكلمتين الرئيسيتين بداية `begin` ونهاية `end` . اكتب العبارة الآتية :

`Close;`



شكل (١-٣) : Hello's form فى Delphi

٧- لقد قمت الآن ببرمجة `procedure` يدعى `OnClick event handler` الخاص بالزر . تأكد من إنهاء العبارة بفصلة منقوطة . قارن النص الموجود فى نافذتك بذلك الموجود فى القائمة (١-١) .

**القائمة (١-١) : برنامج `OnClick event handler` لـ `CloseButton object`**

```
procedure TForm1.CloseButtonClick(Sender: TObject);
begin
    Close;
end;
```

٨- احفظ المشروع ثم قم بتشغيله بضغط الـ `F9` . يمكنك الآن ضغط زر الـ `OK` لإغلاق النافذة . ولأن هذه تعد نافذة البرنامج الرئيسية ، فإن إغلاقها ينهى التطبيق أيضاً .

ان الخطوات السابقة توضح ثلاث جوانب هامة للبرمجة مع `Delphi` :

\* إنك قمت بإضافة `Button component object` على الـ `form` .

\* إنك قمت بتعديل خصائص `object` لتغيير قيم الـ `Name` والـ `Caption` الخاصة به .

\* إنك قمت ببرمجة واحد من الـ `object event` لإداء فعل فى وقت التشغيل استجابة لإختيار الزر .

## الباب الأول : معلومات حول Delphi 4

ان معظم خبرتك للبرمجة مع Delphi تتضمن استخدام component objects, properties, events . لإنشاء التطبيقات ، قم بإضافة component object على ال form ، وقم ببرمجة خصائصهم وبرمجة ال event handler الخاصة بهم . انه أمر غاية في السهولة . وبالمناسبة ، ان واجهة التطبيق لـ Delphi وجميع components تمت كتابتها باستخدام Delphi- وهذا اثبات قوى لما تستطيع تنفيذه مع نظام التطوير المتميز هذا .

### إزالة الأخطاء بجمل البرنامج:

ان واضعى البرامج لا يحبون عملية إزالة الأخطاء . من منا لا يفضل تصميم code جديد عن إصدار الوقت فى إزالة الأخطاء؟ ان Delphi لا يستطيع منع الأخطاء بالطبع ، ولكنه يستطيع معاونتك على تجنب كثير من الأخطاء الشائعة وذلك بإعطائك visual environment تقوم تلقائياً بإنشاء جزء كبير من تطبيقك . للأسف ، مازال هناك فرصة كبيرة لحدوث الخطاء ، ولن اكون صادقاً اذا قلت ان البرمجة مع Delphi تكون خالية من الأخطاء .

اوامر إزالة الأخطاء الخاصة بـ Delphi تنفيذ البرنامج سطر بسطر وراجع قيم ال objects والمتغيرات . وبإبطاء السرعة المعتادة للبرنامج ، يظهر بوضوح سبب المشكلة . تمرن على اوامر إزالة الأخطاء الخاصة بـ Delphi الآن حتى تعرف كيف تستخدمهم حينما تحتاج إليهم - صدقنى ، إنك سوف تحتاج إليهم .

وتعتبر أوامر إزالة الأخطاء الخاصة بـ Delphi مفيدة ايضاً لاستكشاف كيف يعمل البرنامج . عليك استخدامها باستمرار لاختيار الأمثلة الموجودة فى هذا الكتاب وكذلك تلك الموجودة فى دليل Demos التابع لـ Delphi . اتبع الخطوات التالية لتعلم كيفية استخدام بعض خصائص إزالة الأخطاء التابعة لـ Delphi :

١- افتح مشروع Hello اذا لزم الأمر .

٢- اذا لم تكن قد أضفت زر Close على ال form افعل ذلك الآن باتباع التعليمات الموجودة فى الفقرة السابقة .

٣- حدد موضع ال Form1.CloseButtonClick procedure فى ال window editor الخاص بـ Main.pas . قم بادخال العبارة التالية فوق Close



لعرض رسالة والسطران الواقعان بين البداية (begin) والنهاية (end) يجب ان يكونا كالآتى:

```
ShowMessage('Ready to close application');
Close;
```

٤- من الطبيعى ان تقوم بضغط F9 لإجراء عمليتي ال compile، وال link، تشغيل البرنامج. ولكن هذه المرة اضغط F8 لاختيار امر ال Step Over الخاص بـ Delphi من قائمة ال Run. هنا يبدأ البرنامج، ويتوقف عند العبارة الأولى (فى الحقيقة هذا يعتمد على نسخة Delphi التى تستخدمها فمن الممكن ان يتوقف عند unit's initialization، اضغط F8 مرة اخرى اذا حدث هذا) يجب ان ترى نافذة البرنامج. فعندما تظهر، اضغط زر ال Close. وبدلاً من إنهاء البرنامج، فإن Delphi يقوم بايقافه داخل ال Close Button Click procedure وهذا يعد مثال لـ single-stepping (أو الانتقال من سطر إلى سطر)، وهو واحداً من اهم ادوات إزالة الاخطاء المستخدمة اثناء تطوير برنامج.

٥- اضغط F8 مرتين اخرتين لتنتقل الى عبارة ال ShowMessage وتنفيذها. يظهر مربع له زر OK على الشاشة. اضغط هذا الزر. ومرة أخرى، لأنك ضغطت F8 فإن Delphi يقوم بايقاف البرنامج مؤقتاً فى العبارة الواقعة بعد التنفيذ وهى ال Close.

٦- اضغط F9 لتنفيذ عبارة ال Close وقم بتشغيل البرنامج حتى النهاية. بعد ضغط ال F8 لتخطى عبارة أو أكثر، فإنك سوف تستمر غالباً فى تشغيل البرنامج بهذه الطريقة بضغط ال F9. ينتهى البرنامج، ويعود Delphi من نمط ازالة الأخطاء الى نمط التطوير.

وطريقة اخرى للتخطى داخل برنامج هى بتحديد ال breakpoint. عندما يصل البرنامج الى العبارة المشار إليها، فإن Delphi يقوم بإيقاف التطبيق قبل ان يتم تنفيذ العبارة. اتبع هذه الخطوات لتحديد ال breakpoint ولتخطى داخل البرنامج.

١- قم بتحريك المؤشر المضىء الى عبارة ال ShowMessage فى ال Main.pas، واختر Run/Add Breakpoint. اضغط زر ال New فى

## الباب الأول : معلومات حول 4 Delphi

button ، واضغط Enter . والعبرة المشار إليها يتم الآن ابرازها باللون الاحمر (أو اللون الرمادي الداكن على الشاشات احادية اللون). أو، يمكنك توجيه مؤشر الفأرة الى اليسار من العبرة واضغط زر الفأرة لتحديد breakpoint استخدم هذه الطريقة لإنشاء أو إلغاء breakpoints . يمكنك تحديد اكثر من breakpoints واحدة في اماكن مختلفة في البرنامج ، ولكن في هذه التجربة ، فإننا ستقوم بتحديد واحد فقط .

٢- اضغط F9 لتشغيل البرنامج . عندما تضغط زر ال Close يقوم Delphi بإيقاف تنفيذ ال code عند العبرة المشار إليها . اضغط F8 لتخطي هذه العبرة أغلق message dialog بضغط زر OK الخاص به واضغط ال F9 لتشغيل البرنامج حتى اتمامه .

٣- جرب الخطوة رقم ٢ مرة أخرى ، اضغط Alt+F4 لإنهاء البرنامج . ولأن هذا لا يؤدي الى تنفيذ ال CloseButtonClick procedure ، فإن البرنامج ينتهي دون التوقف عند ال breakpoint . ملحوظة : النسخ الاولى من Delphi تعرض ايقونة علامة توقف عندما تقوم بتحريك المؤشر فوق نافذة برنامج متوقف . أما النسخ الأحدث لم تعد تفعل ذلك .

٤- حاول دائماً تشغيل برنامج حتى إنهاء تنفيذه كاملاً وإذا لم تتمكن من فعل هذا ، اختر Run/Program Reset (Ctrl+F2) وابدأ العمل . وهذا يؤدي الى عدم جعل Delphi ينتهي عمليات غلق المشروع بصورة صحيحة وإلغاء ما قد يكون في الذاكرة ، مما يؤدي الى فقد تدريجي للذاكرة المتاحة التي تتطلب منك في النهاية ان تخرج وتعيد البدء في Windows . تلك هي حياة مطور البرامج ، ولكن لحسن الحظ هذه المشكلة نادرة الحدوث .

٥- لإنشاء أو إلغاء breakpoint ، حرك مؤشر الفأرة الى اقصى اليسار من العبرة المشار إليها واضغط الزر الايسر مرة . تلك هي اسهل طريقة لإنشاء أو إلغاء ال breakpoint يمكنك ايضاً استخدام امر View/Breakpoints لإضافة ، أو لحذف ، أو إبطال أو أداء عمليات اخرى على ال breakpoint . اختر هذا الامر الآن واضغط زر الفأرة الايمن في نافذة قائمة ال Breakpoint للحصول على قائمة من الاوامر المتاحة . قم بإزالة ال breakpoint من عبرة ال ShowMessage الآن وذلك باختيار امر ال Delphi من القائمة .

وسأقدم لك اوامر اخرى لإزالة الاخطاء خاصة بـ Delphi وذلك فى الاوقات المناسبة خلال هذا الكتاب. ولكن لا تنتظرني لاقتراح عليك تخطى عبارات برنامج وذلك بضغط الـ F8 وتحديد الـ breakpoint. عليك استخدام خصائص إزالة الاخطاء فى Delphi باستمرار وذلك لتصحيح الأمثلة فى هذا الكتاب. وانك سوف تنبهر بما ستعرفه فقط بإبطاء سرعة تنفيذ الـ Code واختيار حالة البرنامج فى مواضع استراتيجية.

### :(Code Insight)

من الاشياء الجديدة والتي تم تقديمها فى Delphi 3 هو ان الـ code insights تقوم تلقائياً بتكملة code اثناء قيامك بعملية الكتابة على الحاسب، كما انها توفر المعلومات حول component libraries و functions و procedures التابعين لـ Delphi باستخدام الـ code insights، يمكنك إنشاء مجموعة من عناصر البرمجة العامة مثل الـ procedure والـ loop يمكنك حينئذ ملئ المجموعة لحفظ وقت الكتابة. يمكنك أيضاً الحصول على الـ Delphi's code editor الذى يعرض انواع الـ parameters لهذه الـ procedures و functions ويوضح أيضاً خصائص و methods و component objects ان توفر هذه المعلومات فى code editor يساعدك على اختصار الوقت المبذول فى البحث من خلال الـ online help الخاصة بـ Delphi وهذا اذا ما عرفت كيفية استخدام code insights جيداً. وهذه الفقرة سوف تساعدك.

فى بعض الاحيان، اجد ان الـ code insights تشكل مشكلة اكثر من فوائدها- وخاصة على الحاسبات الاقل سرعة والتي تتوقف لفترة طويلة قبل عرض الـ on line help الخاصة بالمضمون. على الناحية الأخرى، إننى اجد الـ code insights غير ذى قيمة كلما استخدمتها. على سبيل المثال، لإنشاء procedure جديد، إننى اقوم بكتابة كلمة procedure واضغط Ctrl+J، وهذا يؤدى الى إنشاء procedure كامل والذي يمكننى من ملأه. وباستخدام انواع اخرى من code insight - code completion، code parameter. وفى الفقرات التالية، أوضح لك أمثلة لكل تلك التقنيات.

من المحتمل انك تريد ان تجعل الـ code insights ملائمة لإسلوب برمجتك، ولكن قبل ان تقضى وقتاً طويلاً فى صنع هذا، من الافضل ان تتأقلم

## الباب الأول : معلومات حول Delphi 4

على المواضيع الموجودة حسب نظام البرنامج وخصائص code insights الاربعة الاساسية هي :

Code Templates \*

Code Completion \*

Code Parameters\*

Tool-tip Expression Evaluation \*

والفقرات التالية تعطينا ارشادات لاستخدام code insights وكذلك مقترحات لمعرفة كيفية العمل بادوات البرمجة القيمة هذه .

### Code templates

اضغط الـ Ctrl+J فى أى وقت وانت تستخدم code editor لعرض قائمة لـ code templates . اختر template الذى تريد ، ويقوم Delphi تلقائياً بإضافة الـ template فى source code فى الاتجاه الحالى للمؤشر ولان هناك مؤشرات عديدة ، يمكنك اختصار الوقت بكتابة الجزء الأول من templates الذى تريد قبل إضغط الـ Ctrl+J . على سبيل المثال ، لإدخال عبارة if فى الـ code ، اكتب كلمة if واضغط Ctrl+J . يضع Delphi أربع من الـ (code templates) الممكنة الآتية فى قائمة :

if statement	ifb
if then (no begin/end) else (no begin/end)	ife
if then else	ifeb
if (no begin/end)	ifs

يصف العمود الأول نوع الـ statement template . ويعطى العمود الأخير اسم الـ template مختصراً ، والذى يمكنك استخدامه لعرض template اذا لزم الأمر . اذا لم تكن تريد عرض template يمكنك تجاهل اسماء المختصرة .

إنك تعرف المزيد عن الـ code template باستخدامها أكثر من مجرد القراءة عنها هنا لذا لن احاول ان اذكر الأنواع الكثيرة المتاحة أقض بعض الدقائق فى اختيار الـ template مختلفة للعبارات التى تنوى استخدامها فى الغالب . اننى اجد ان

بعض الـ template أكثر فائدة من غيرها . على سبيل المثال ، إننى استغرق وقتاً أطول لتحريك المؤشر حول if-then-else أكثر من لو أننى قمت بكتابة الكلمات الأساسية مباشرة - ولكننى اكتف بسرعة ولكننى أجد template أخرى أكثر فائدة ، خاصة لأننى أيضاً استخدم الـ C++ ولغات برمجة أخرى . على سبيل المثال ، ان code templates تذكرنى ان الـ Object Pascal exception تستخدم الكلمتين الأساسيتين try و except ، بينما لغات أخرى قد تستخدم try و catch . ان الـ Code insights تساعدنى على التنبيه إلى جميع الـ Object Pascal Syntax ، وتخفف من أخطاء أثناء عملية الـ compilation .

**فكرة:** لا تضع مسافة بعد الجزء الأول من الـ code template . اكتب فقط الكلمة الأولى من العبارة - على سبيل المثال if ، أو for أو while - ثم اضغط Ctrl+J . يضيف Delphi مسافات حسب تصميم الـ template .

بعد أن تعود على استخدام الـ code template ، قد تريد أن تقوم بعرض المجموعة المخزنة وقد تريد أيضاً أن تنشئ الـ template الخاصة بك . ولأغراض العرض ، سوف تقوم بإنشاء procedure template جديد والذي يسبقه تعليق يمكنك أن تملأه لوصف الـ code template ولتفعل ذلك اتبع هذه الخطوات :

١ - اختر ToolsEnvironment Options واضغط Code Insight page tab . اضغط زر الـ Add ، وأدخل procedure كاسم المختصر لـ template ، و "procedure (commented)" كوصفه .

٢ - اضغط داخل editor window الواقعة بعد علامة الـ Code . أدخل السطور التالية أو استخدم تصميمك الخاص (يشير vertical bar الى المكان الذى تريد أن يضع فيه Delphi مؤشر النص بعد إدخال template فى code البرنامج) .

```
{ Purpose: |
  Date   : 00/00/00
  Author : Tom Duck
}
procedure ();
```

begin

end;

يمكنك الآن إضافة template آخر أو يمكنك عرض الـ template الموجودة .  
اضغط OK لحفظ تغييراتك . لاستخدام الـ template ، اكتب P ثم اضغط  
Ctrl+J . هذا يقدم لك قائمة بأسماء جميع الـ template التي تبدأ بحرف الـ P .  
اختر الـ template الجديد المسمى procedureC لإدخال الـ template الى الـ code editor .

**ملحوظة:** اذا كان هناك template واحد عندما تقوم بضغط الـ Ctrl+J ، يقوم Delphi مباشرة بإدخال الـ template الى الـ code editor .  
باتباع الخطوات السابقة ، عليك الآن الاختيار من بين اثنين من الـ template .  
لكلمة الـ procedure الرئيسية . وللعودة للوضع الافتراضى ، افتح الـ Code Insight dialog ، اختر الـ template الجديدة (procedureC) ، واضغط زر الـ Delete .  
يمكنك الآن أن تكتب الـ P وتتبعها بـ Ctrl+J لإدخال الـ template الـ procedure مباشرة الى الـ Code . وبالمثل ، يمكنك كتابة F يتبعها Ctrl+J لإضافة الـ function template .

Note

### :Code completion

إن إدارة الـ code completion الخاصة بـ Delphi أيضاً تقوم بإدخال النص تلقائياً إلى الـ code editor ، ولكنها تختلف عن الـ code templates بأنها تعطى الـ methods والخصائص والـ events والـ arguments . إن هذه الخاصية توفر الوقت وخاصة فى أنها تساعد على منع أخطاء الـ compiler أكثر من مجرد توفير وقت الكتابة . على سبيل المثال ، يمكنك وضع كل الـ methods الممكنة لأى VCL class object فى قائمة . وبالإختيار من هذه القائمة ، فأنت أقل عرضة للوقوع فى خطأ فى التراكييب ، أو تعيين قيمة خاطئة أو محاولة استدعاء الـ methods غير موجودة .


وإننى أجد code completion مفيد بشكل خاص فى برمجة الـ objects من خلال الـ source code فبدلاً من البحث عن أسماء الخصائص والـ methods

عن طريق online help الخاصة بـ Delphi، فإنني اكتب جزء من الـ code وانتظر لحظة حتى يظهر الـ code completion insight عندئذ أقوم بإختيار الطريقة أو أى بند آخر أريده وأستمر فى الكتابة قدر ما يوفر وقت البحث .

اتبع الخطوات التالية لتجربة code completion ولترى كيف تختلف عن الـ code templates :

١- إبدأ تطبيق جديد بإختيار FileNew Application . يمكنك تسمية وحفظ التطبيق اذا أردت ذلك، أو يمكنك استخدام الاسماء الموجودة .

٢- أضف الـ Button component من Standard component palette الى الـ form الخاصة بالتطبيق . اضغط مرتين الى الـ button object لتكوين event handler .

**فكرة:** هذه هى أسرع طريقة لإنشاء لـ event handler object ما ، بالرغم من أنه بإمكانك استخدام نافذة الـ Object Inspector كما هو موضح بالنسبة لتطبيق Hello فى هذا الباب . 

٣- يجب وضع المؤشر بين كلمة البداية الرئيسية وكلمة النهاية الرئيسية فى الـ Button1Click procedure استخدم الآن الـ code completion لتغيير الـ button's label عند ضغطه . أولاً ، اكتب الآتى :

Button1.

ثم انتظر لحظة . لا تضع مسافة بعد النقطة وسوف ترى اقتراحات code completion لهذا النوع من الـ object . (اذا اختفت المقترحات ، امسح النقطة واعد كتابتها حتى تصل الى code completion) يقوم Delphi باعطاء قائمة لكل الخصائص و constructors و procedures و functions لهذا النوع من الـ object's . فى هذه الحالة ، نود ان نعين String جديد لخاصية الـ Caption الخاصة بالـ object أبحث عن كلمة Caption فى القائمة ، واضغطها مرتين لإضافة الـ Caption فى الـ source code . اكتب String جديد لتحديده للـ Caption فى الـ زر . على سبيل المثال ، قم بكتابة هذا السطر ليكون كالآتى :

Button1.Caption := 'Ouch!';

## الباب الأول : معلومات حول Delphi 4

٤- يمكنك الآن تشغيل البرنامج وضغط الزر ، مما يغير نصه الى "Ouch!" .

**فكرة:** لتغيير كم الوقت الذى عليك انتظاره حتى تظهر نافذة Code Insight page tab ، استخدم ال Tools|Environment Options..... اختر Code Insight page tab واسحب Delay slider لتحديد القيمة التى تريدها . وفى desktop system فإننى استخدم اسرع توقيت وهى نصف ثانية . اما على desktop system ، فإننى ازيد هذه القيمة فيتظهر النافذة اذا انتظرت ثانية أو أكثر بهذه الطريقة ، يمكننى الاستمرار فى الكتابة اذا فضلت عدم رؤية النافذة ، والتى تأخذ لحظة متى تشكل . جرب مواضيع أخرى لترى أى توقيت يعمل بشكل افضل على نظامك .

وطريقة أخرى لاستخدام code completion هى ان تضغط قضيب المسافة Ctrl+. هذا يجبر ال code completion على العمل ، والتى تعطى قائمة بالإشياء المتاحة لهذا الموضع فى source code . وعلى النقيض من documentation التى قد تقرأها ، هذه الخاصية لا تعطى قائمة بالقيم الصحيحة لتعيين عبارات - إنها تقوم قائمة بكل الأشياء الموجودة ولذا يمكن تخصيصها أو استخدامها بفاعلية فى هذا المكان فى البرامج . وقد يفيد هذا فى عبارات التعيين ، ولكنه مفيد أيضاً فى جوانب أخرى ، مثل إيجاد اسم glabal declaration .

على سبيل المثال ، اننى اجد هذا النوع الثانى من ال code completion غاية فى الافادة فى الاختيار من بين ال local and global variable المتعددة للبرنامج لبدء عبارة تعيين ، فإننى اضغط قضيب المسافة Ctrl+. ثم اختار اسم متغير من القائمة الناتجة . وهذا يمكننى من استخدام اسماء متغيرات وصفية وطويلة مثل CountOfAliveObjects ، واختيارها من القائمة بدلاً من كتابة هذه الاسماء مراراً .

**ملحوظة:** تذكر انه بإمكانك استخدام ال code completion بطريقتين . الاولى ، اكتب اسم object ثم ضع نقطة ، ثم انتظر لحظة لتظهر قائمة بخصائص ال object و methods . الثانية ، اضغط قضيب المسافة Ctrl+. فى أى وقت للحصول على قائمة بال objects المتاحة فى هذا المكان . source code

Note



ان ال code completion يتطلب ان تكون source code صالحة من ناحية Syntax على الاقل فى مكان الادخال . اذا تلقيت رسالة " غير قادر على استدعاء ال code completion " نتيجة اخطاء فى ال source code ، تأكد من انك لم تنس كلمة اساسية أو أى عنصر هام آخر . لرؤية هذا الخطأ ، حرك المؤشر بعد نهاية unit's source code (بعد نقطة النهاية) واضغط قضيب المسافة +Ctrl . فى الحقيقة قد لا يكون هناك اية اخطاء فى هذه الحالة - المسافة الواقعة بعد نهاية ال unit تعتبر خارج نطاق code ، وهذا هو سبب تلقيك هذه الرسالة).

امامك طريقتين لتنظيم ال code completion يمكنك تغيير الوقت الذى تستغرقه نافذة completion للظهور ، أو يمكنك إيقاف code completion استخدم ال.....Tools|Environment Options واختصر ال Code Insight لعمل أى من التعديلات قد تريد إيقاف code completion على الحاسبات البطيئة للغاية ، والتى قد تستغرق وقت طويل لإظهار نافذة completion . فى هذه الحالة ، فعليك ان تستخدم online help و documentation لترى الخصائص و methods و objects .

### :Code parameters

واليوم الثالث من ال code insight . جعل من المستحيل يرفض ما قد تحدده كقيمة بالنسبة لك methods الخاصة بال object وهى احد اخطاء البرمجة الأكثر شيوعاً . مع هذه الخاصية ، يقدم نافذة قائمة بكل القيم المتغيرة المطلوبة بالنسبة لل method التى يستدعيها code . فانت ببساطة تستعين بهذه المعلومات لتحديد القيم الصحيحة لل parameters الخاصة بكل من ال parameters وال functions .

التجربة ستظهر لك كم هى قيمة ال code parameter هذه . واننى استخدم ايضاً code parameters لتوضيح كيفية استخدام هذين النوعين من code insights معاً ، وهو ما سنفعله دائماً . اتبع هذه الخطوات :

١ - ابدأ تطبيق جديد وأضف Button object على ال form اضغط مرتين الزر لإنشاء ال event handler .

٢ - بين كلمة البداية الرئيسية وكلية النهاية الرئيسية ، اكتب Button1 (لاتنسى ان تضع نقطة) وانتظر لحظة . اكتب sc لإظهار ال ScaleBy method فى

## الباب الأول : معلومات حول Delphi 4

نافذة code parameters ، واضغط Enter لإضافة هذا method داخل source code (تذكر هذه الطريقة - عندما تظهر نافذة code parameters ، اكتب الأحرف الأولى من method أو الخاصة التي تريدها ، ثم اضغط Enter لإدخالها في البرنامج .

٣- اضع اقواس الى عبارة الـ `Button1.ScaleBy` التي تم تشكيلها جزئياً ثم انتظر لحظة يجب ان ترى القائمة التالية من انواع الـ parameters يمكن تحديدها للـ `ScaleBy` method .

M: Integer; D: Integer

يظهر هذا النص في مربع تحت مؤشر النص مباشرة . لاحظ ان الـ parameter الأولى مكتوبة بخط سميك - هذا يشير الى parameter الذي من المتوقع ان تكتب بعد ذلك . ادخل 4 وفصله ، ومرة أخرى ، انتظر لحظة . والآن يصبح الـ parameter D مكتوبة بخط سميك مشيراً الى ان هناك قيمة رقمية متوقعة لهذا الـ parameter . اكتب فى ، اغلق القرص ثم ضع فصلة منقوطة لإنهاء العبارة .

عندما تضع الاقواس ، تختفى النافذة الخاصة بقيم الـ parameter ، مما يشير الى انك قد قدمت كل الارقام المطلوبة . فكرة : تذكر ان تضع فصلة ، وليس فصلة منقوطة ، بين القيم الرقمية الحقيقية - وهى معلومة يعرفها واضعى برامج الـ Pascal المتمرسين ولكن يجهلها المبتدئين . والعبارة التامة يجب ان تبدو كالتالى :

`Button1.ScaleBy(4, 3);`

٤- يمكنك الآن ان تضغط F9 وتشغل البرنامج . ان الزر يكبر اكثر فى كل مرة تضغط عليه ، وكأنه احد الاشياء الموجودة فى اليس فى بلاد العجائب . وهو يصبح هكذا لان الـ `ScaleBy` method يحدد الـ VCL object بـ M مقسومة على D ، وفى هذه الحالة 4/3 (أو ٤/٣) أو حوالى ١,٣٣ من حجمه .

**فكرة:** ان النافذة الخاصة بقيم الـ parameters قد تستغرق دقيقة أو اثنتين لتظهر ، خاصة للـ parameters التي لها أكثر من arguments مختلفة كما هو الحال فى الـ code completion ، يمكنك تغيير وقت التأخير لهذا الخيار أو يمكنك ابطاله بواسطة امر الـ Tools|Environment Options...

## :Tool-tip expression evaluation

ان code insight هذا يبدو أكثر غرابة عما هو فى حقيقته، ولكن لا تستطيع القول بأنه ليس فائدة. فى الواقع، قد تكون هذه الخاصية هى التى ستستخدمها فى معظم الاوقات فى اختيار الcode الخاص بك. لاستخدام tool-tip expression evaluation، يجب ان تقوم بتشغيل برنامجك فى حالة debug ويمكنك فعل هذا باكثر من وسيلة، ولكن اشهر وسيلتين هما ال breakpoint أو بتشغيل code فى نمط تنفيذ الcode خطوة خطوة بضغط F8.

عندما تفعل هذا ويتوقف البرنامج فى مكان ما اثناء تنفيذه، حرك المؤشر فوق أى متغير لترى نافذة توضح قيمته. ويعتبر هذا ذو منفعة عظيمة خاصة فى فحص المتغيرات فى حالة debugging. ويجب ان يكون المتغير ظاهراً فى هذه النقطة من تنفيذ البرنامج ان المتغيرات التى تتصف بصفة ال local بالنسبة لل methods التى يتوقف عندها البرنامج تعتبر غير قابلة للفحص (فى الحقيقة) هذه المتغيرات غير موجودة فى الذاكرة). وكمثال على كيفية عمل هذه الخاصية، اتبع الخطوات التالية:

١- قم بانشاء تطبيق جديد، ادخل Button Button، اضغطه مرتين، وادخل العبارة التالية. فى Button1Click method:

Button1.Caption := 'Ouch!';

٢- إنها نفس العبارة التى قمت بكتابتها لمثال code completion فى هذا الفصل. قبل تشغيل التطبيق، ادخل breakpoint على العبارة السابقة. قم بهذا بالضغط الى اليسار من الخط، أو باستخدام امر ال Run!Add Breakpoint....

٣- اضغط F9 لتشغيل التطبيق ثم اضغط الزر هذا يؤدى الى توقف البرنامج عند العبارة التى حددت بها ال breakpoint. ولان البرنامج قد توقف فى حالة debug، يمكنك الان استخدام tool-tip expression evaluation لاختبار المتغيرات. حرك المؤشر فوق كلمة Caption. انتظر لحظة، ويقوم Delphi بعرض القيمة التى تم تعيينها حالياً لهذه الخاصية. وفى مربع صغير ترى الآتى:

Button1.Caption = 'Button1';

## الباب الاول : معلومات حول Delphi 4

٤- قد تم تحديد ال Caption لهذا ال String لان البرنامج لم يتم بعد بتنفيذ العبارة في ال breakpoint ، قم بتحريك المؤشر فوق ال Caption ، والذي يظهر الآن القيمة الجديدة التي تم تعيينها في آخر مرة تم فيها تنفيذ هذه العبارة .

كما هو الحال في code completion و code parameters ، يمكنك تغيير وقت التأخير الخاص tool-tip expression evaluation ، أو إيقاف هذه الخاصية باستخدام امر ال Tools|Environment Options . على الرغم من ذلك ، لا اعتقد انك تريد إيقاف مثل هذه الخاصية المفيدة .

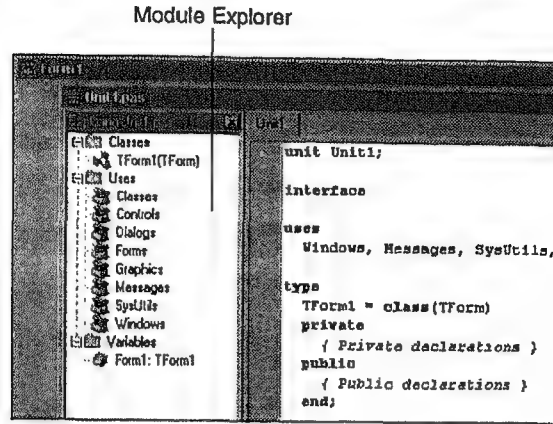
**Tip فكرة:** Tool-tip evaluation تعمل بصورة افضل اذا تم إغلاق ال compiler optimizations ، لأنه عند تشغيل ال optimization ، قد يقوم ال compiler باعادة ترتيب ال code ، بطريقة ما بحيث يصبح وضع المتغيرات في الذاكرة امر صعب أو مستحيل لـ debugger . ولإيقاف ال optimization عليك اختيار ال Project|Options... واختر Compiler page tab . وتحت باب Code generation ، امسح مربع خيار ال Optimization . ويجعل هذا يحدث تلقائياً ، قم بتشغيل مربع خيار ال Default dialog قبل ان تضغط OK . تأكد من تشغيل ال optimization قبل إجراء عملية compiling النهائية .

## : Module Explorer

ان Module Explorer ، وهو جديد في Delphi 4 ، يبسط لك الطريق امام ال source code files . ان ال Module Explorer يقدم outline خاص بكافة عناصر برنامجك . ان هذا ال object يعمل على document the source ويساعدك ايضاً في ايجاد ال declarations محددة . يوضح الشكل (١-٤) ال Module Explorer من شكل (١-١) .

يظهر ال Module Explorer الى اليسار من code editor . وهو يعرض tree outline ، خاص بكل من classes ، properties ، methods ، global variables ، و global routines .

ويقدم ال Module Explorer ايضاً قائمة باسماء ال units المستخدمة من قبل ال module .



شكل (١-٤): ال Module Explorer

للربط بين ال Module Explorer و code editor windows ، اضغط **Ctrl+Shift+E**. قم بإختيار أى عنصر فى tree-outline object ، ويقوم Delphi بالانتقال مباشرة الى مكان تعريف الرمز فى code editor اننى اجد ان هذا مفيد جداً بالنسبة لتحديد موضع تنفيذ methods يكون قد تم تعريفه فى الجزء interface على سبيل المثال تعريف method من خلال class .

ان Module Explorer يساعد على البحث بصورة تصاعدية اكتب الاحرف الأولى من الرمز الذى تريد ايجاده . اذا كان هذا الرمز معرف من خلال البرنامج ، فإن المؤشر يذهب تلقائياً الى مكان فى code editor .

### : Docking and undocking

يمكنك إجراء عمليتى Docking و docking ل Module Explorer . لإجراء عملية undocking اضغط واسحب الحد العلوى للنافذة . والذى يمكنك ان تسحبه الى أى موضع . اطلق زر الفأرة لعرض ال Module Explorer فى نافذة منفصلة . وقد تقوم بتنفيذ هذه الحركة فى low-resolution display ، فى هذه الحالة ، قد يجعل ال Module Explorer الميعارى ال code editor صغيراً جداً . وفى حاسبى الذى يعرض ٧٦٨×١٠٢٤ ، افضل ان اترك ال Module Explorer متصلاً بموضعه الاصلى

## الباب الأول : معلومات حول Delphi 4

بعملية الـ undocking يمكنك اظهار نافذة الـ Module Explorer باختيار View|Module Explorer . اضغط الـ Ctrl+Shift+E عند تعاملك مع الـ code editor، هذا سوف يؤدي الى جلب الـ Module Explorer امام النوافذ الأخرى . ولأجراء عملية Docking لـ Module Explorer وإرجاعه الى مكانه الاصلى مرة أخرى ، اضغط واسحب حد النافذة الى اليسار من code editor . استمر فى السحب حتى يتغير حجم الـ outline من الحجم الحالى الى الحجم الأقل أترك زر الفأرة حتى تمام عملية الـ docking .

يمكنك إجراء عملية الـ Docking لـ Module Explorer الى اليمين من نافذة الـ code editor تحكم بالحد الأعلى الخاص به بواسطة الفأرة واسحب نافذة الـ Module Explorer الى جانب الآخر من editor . استمر فى السحب حتى يتغير حد حجم الـ outline من الحجم الحالى الى الأقل حجماً ثم أترك زر الفأرة .

**Tip** . **هكرة:** الـ code editor عندما يكون maximumized فقد يقل حجم الـ outline ويمكنك ترك زر الفأرة فى هذا الوقت لاستكمال عملية الـ docking .

### Class Completion

إذا قمت بالبرمجة مع Delphi ، فإنك من المحتمل إرتكابك للعديد من الاخطاء فى تعريف خاص بالـ method لأى class أو عند محاولة كتابتها بصورة صحيحة .

عندما تفعل هذا ، فتكون بالتأكيد قد نسيت ان تصنع اسم الـ class . والنقطة التى تليه ، أو كررت التعريف مرة أخرى ، يسبب تلك الأخطاء قد تظهر عدة رسائل توضح وجود خطأ منهما .

وباستخدام خاصية الـ Module Explorer's class completion والتى قد تعتبرها نوع من اداة من أدوات الـ code insight الجديدة - فإن code editor يمكنه إنشاء method shell لأى تعريف جديد . ولتجربة هذه الخاصية ، ابدأ تطبيق جديد ، ثم اتبع هذه الخطوات :

١ - أضف Button object على الـ main form الرئيسية للبرنامج .

٢- اضغط مرتين Button1 لإنشاء OnClick event الخاص لل objects.

٣- اضغط بالفأرة TForm1 فى ال Module Explorer . هذا يؤدى الى تحريك المؤشر الى تعريف TForm1 class . حدد موضع الكلمة الاساسية الخاصة وتحتها مباشرة اكتب تعريف procedure التالى ، والذي يوضح ان ال Subroutine كعضو فى TForm1 class :

```
procedure Subroutine;
```

٤- لتكملة ال code ، عليك ان تحدد ال implementation التى تتبع لها ال method . ضع المؤشر فى أى مكان من الجزء الخاص بتعريف TForm1 class ، ثم اضغط Ctrl+Shift+C . يقوم Delphi بانتاج قائمة بكافة العناصر التى تكون معرفة دون التعامل معها .

٥- ومع ال one unimplemented method فإنه يتم وضع المؤشرين البداية الرئيسية لل Subroutine والنهاية الرئيسية لها . أضف العبارتان لاتمام ال method كما يلى :

```
procedure TForm1.Subroutine;
begin
    ShowMessage('Ready to exit');
    Close;
end;
```

٦- حدد موضع OnClick event handler من الخطوة رقم ٢ . لعمل هذا بسرعة باستخدام ال Module Explorer ، اضغط Button1Click under TForm1's methods .

٧- اكتب العبارة التالية بين كلمة البداية الرئيسية وكلمة النهاية الرئيسية والاسلوب التام يجب ان يبدو كالاتى :

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Subroutine;
end;
```

## الباب الأول : معلومات حول Delphi 4

٨- اضغط F9 لتشغيل البرنامج. عندما تضغط الزر، يقوم OnClick event handler بإستدعاء ال Subroutine، والذي يعرض رسالة وينتهي البرنامج.

ويعمل ال Class completion ايضاً بشكل عكسي. قم بإضافة procedure أو function فى الجزء implementation section قم بتحريك المؤشر الى أى مكان داخل ال procedure وال function، ثم اضغط Ctrl+Shift+C لإنشاء تعريف لهذا ال method فى ال class المشار إليها. لتجربة هذا، عليك ان تمحو تعريف ال class من القطاع ال private لل TForm1. استخدم ال Module Explorer للوصول الى تنفيذ ال Subroutine واضغط Ctrl+Shift+C. اذا لم يظهر أى تعريف لل method، يقوم Delphi بإضافة واحداً فى القطاع ال private لل class. (يمكنك نقل هذا التعريف الى إحدى الجزئين الآخرين ال protected أو public اذا لزم الأمر.

### : Module navigation

ويقدم ال Module Explorer طريقة أخرى لربط ال code editor بين class declaration من خلال ال class التابع له وبقيّة عناصره. ومع وضع المؤشر على أى method declaration اضغط سهم ال Ctrl+Shift+Down أو سهم ال Ctrl+Shift+Up للقفز الى ال method's implementation (ليس من المهم أى مفتاح يهم سوف تضغط لان الخاصية هى الربط بين شيئين، والاتجاه لا يهم).

واحدى الملاحظات فى ملف الخاص بـ Delphi help تشير الى انه بإمكانك استخدام هذه المفاتيح للربط بين الجزء interface والجزء implementation فى ال unit، ولكن هذا لا يبدولى فعلاً. قد يكون هذا ثابتاً فى نسخة Delphi الخاصة بك.

### الملفات وامتدادات اسم الملف:

ان الفقرات التالية تضيف محتويات انواع عديدة من الملفات التى يقوم Delphi بإنشاءها للتطبيق. وسوف اوضح لك أى الملفات يمكنك حذفها بسلام من دليل المشروع لتوفير مساحة من القرص.



## ملفات Source code :

عندما تقوم بإضافة component على ال form، وعندما تقوم بتعديل event والخصائص لل form، أو component، فإن Delphi يكتب نص برنامجك بال Object Pascal. وهذا النص يسمى source code البرنامج. فى أى وقت أثناء تطوير تطبيق ما، يمكنك استعراض ملفات ال source code للبرنامج، ويمكنك طباعتها كنسخ احتياطية من الورق.

ان واضعوا برامج ال Pascal ذو الخبرة قد يريدون تعديل ال source code الخاصة بتطبيقاتهم. بغض النظر عن خبرتك، عليك ان تقوم بإدخال برمجة جديدة بحذر شديد.

ولكنك لا تستطيع فهم كل تغيير تقوم به. وللحصول على افضل نتائج، اتبع المقترحات الموجودة فى هذا الكتاب لإنشاء التطبيقات. وبمرور الوقت، سوف تصبح القواعد الخاصة بما تستطيع تغييره وما لا تستطيع تغييره واضحة لديك.

اتبع هذه الخطوات لاختيار source code لتطبيق مثال Hello ولتعتاد اكثر على البرمجة والملفات التى يولدها Delphi :

١- اذا لم يكن مشروع Hello مفتوحاً، اختر File\Open.... لفتح ملف مشروع Hello.dpr.

٢- اختر Select View\Units (Ctrl+F12) أو اضغط زر السرعة Select-unit-from-list، واختر Main from قائمة modules. ويعرض هذا source code فى Delphi editor. اضغط Enter أو اضغط OK لإغلاق unit dialog وتوضح القائمة (١-٢) النص الذى تراه على الشاشة. ويتم تخزين نفس هذا النص على القرص المدمج على أنه Main.pas فى دليل ال Source\Hello.

٣- اختر View\Units مرة أخرى ولكن هذه المرة اختر Hello module. وتظهر نافذة ال editor الآن ملفين- اختر فيما بينهما بضغط ال page tab فى أعلى النافذة. توضح القائمة (١-٣) نص ملف المشروع، الذى تم تخزينه على القرص المدمج على أنه Hello.dpr فى دليل ال Source\Hello.

القائمة (١-٢) : Hello\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    CloseButton: TButton;
    procedure CloseButtonClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.CloseButtonClick(Sender: TObject);
begin
  Close;
end;

end.
```

### القائمة (٢-١) Hello\Hello.dpr

```
program Hello;

uses
  Forms,
  Main in 'Main.pas' {Form1};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

**فكرة:** لإغلاق ملفات فردية فى ال editor، تعامل مع هذه الصفحة واضغط **Ctrl+F4**، أو اضغط زر الفأرة الأيمن واختر **Close Page**. إن هذا ببساطة يغلق الصفحة فى editor- إنه لا يحذف فى الملف من القرص. ولإعادة فتح ملف **source code** خاص بـ **unit** ما، اختر اسمه باستخدام أمر ال **ViewUnits**.

إن ال **source code** لمشروع **Delphi** يتم تخزينها فى إثنين من ملفات النص على الأقل فى هذه الحالة، ال **Main.pas** وال **Hello.dpr**. ويعتبر ملف ال **Main.pas** فى القائمة (٢-١) مثال لـ **unit**، والتي تحتوى على برمجة ال **form** الرئيسية لبرنامج **Hello** وكل التطبيقات الموجودة فى هذا الكتاب لها على الأقل **unit** واحدة، ولكن المشروعات المعقدة من الممكن أن يكون لها العديد من ال **units** كل منها قد تم تخزينه فى ملف **pas**. منفصل.

وملف **second source** الثانى فى تطبيق **Delphi** النموذجى يحتوى على **source code** لمشروع بأكمله. على سبيل المثال، يوضح **Hello.dpr** فى القائمة (٣-١) ال **source code** لمشروع **Hello**. ولكل تطبيق ملف مشروع واحد فقط. ولطباعة الملف الذى تم اختياره حالياً فى نافذة **editor**، اختر **File|Print...** يمكنك الاختيار من بين العديد من الأوامر الموجودة فى **Print Selection dialog**

## الباب الأول : معلومات حول Delphi 4

النتائج. على سبيل المثال، قم بتحديد خيار الـ Line numbers لإعطاء رقم لكل سطر برمجة، أو قم بتحديد خيار الـ Header/page number للحصول على المظهر التام. عليك بتشغيل خيار الطبع Syntax لطباعة الكلمات الرئيسية بخط سميك ولاستخدام الخط المائل underlining لعناصر البرامج الأخرى. قم بإيقاف طباعة الـ Syntax للحصول على طباعة سريعة على الطابعات البطيئة أو إذا كنت تفضل الحصول على مظهر واضح نسبياً في القوائم.

**ملحوظة:** إن النسخ الحديثة من Delphi تتضمن خياراً للطبع بالألوان. وتوضح القائمة الناتجة الكلمات الرئيسية وعناصر البرنامج الأخرى باستخدام إبراز تكوينات الجمل، وهو ما يشبه ما تراه في نافذة الـ Delphi editor الخاصة، وأنت أيضاً تحتاج إلى استخدام خيارات الـ Tools|Environment Options... لإختيار ترتيب اللون البديل حسب نظام البرنامج- والمواضع الأخرى مثل Classic، على سبيل المثال، تبدو أفضل على الشاشة منها على الورق. قم بإجراء التجربة على ملفات إختيارية حتى تحصل على المنظر الذي تريده. للأسف، عليك أن تقوم بالفعل بطباعة صفحة لتغيير خيارات الطبع- لتغييرها وإلغاء إعادة الطبع إلى المواضع القديمة.

### ملفات الـ Unit:

والآن، فلتنظر من قريب إلى source code التي يولدها Delphi لمشروع ما، بداية من form unit الرئيسية، Main.pas [القائمة (١-٢)]. إن السطر الأول في هذه الحالة هو الـ Main. ويعطى Delphi unit نفس الاسم الذي تحدده أنت عندما تقوم بحفظ unit file للمشروع.

ننتقل إلى التعامل مع الجزء interface والتي تجعل العناصر التي تلي التعريف متاحة لوحدة أخرى ليستخدم هذا الاسم. ويحتوي جزء الـ interface على public declarations للـ unit. والعناصر التي ليست موجودة في الجزء interface والتي يتم منع استخدامها مع العناصر الأخرى.

وغالباً ما تستخدم الـ units، عناصر unit أخرى. على سبيل المثال، إن Main تستخدم units أخرى عديدة، كلها موجودة في قائمة تحت الجزء uses في

أعلى ال Unit على سبيل المثال Windows ، Messages ، SysUtils ، وغيرها  
وهي توفر بيانات و code الخاص بـ Windows API ، أو الاستعمال لبيانات  
و code برنامج آخر ، أو Delphi component . وبإمكانك أيضاً إنشاء units  
خاصة بك واستخدامها . على سبيل المثال ، بإمكانك إنشاء unit ذات بيانات  
و code يمكن التشارك فيها مع أجزاء مختلفة من البرنامج .

في ال Main.pas نأتى إلى قطاع type declaration ، وعندما نضع كلمة  
تعريف العنصر فإننا نريد تحديد طبيعة العنصر الذى يمكن إنشاؤه فى الذاكرة . على  
سبيل المثال ، إن ال Integer يعد نوع بيانات يمثل الاعداد الصحيحة . ولاستخدام ال  
Integer ، عليك إنشاء متغير أو أكثر من هذا النوع يمكنه تخزين قيم معينة مثل  
٣٢١ . وتشمل أنواع بيانات ال Pascal الأخرى strings ، characters ،  
records و ال floating-point سوف أوضح الكثير عن هذه الأنواع وصادفت  
مثالاً عن هذا فى الأمثلة لهذا الكتاب .

ونوع البيانات يمكن أن يكون أيضاً class بمعنى object-oriented  
structure يستخدم كثيراً فى تطبيقات Delphi . وتصف ال class خصائص  
ال object الذى يمكن إنشاؤه فى الذاكرة . وقد تعرف class encapsulates .  
ويمكن لل class أن تتعامل مع class members أخرى ، على سبيل المثال ، إن  
TForm1 class فى ال Main.pas تتعامل مع TForm1 class ، وهى class  
يقدمها Delphi لإنشاء forms تسمى بالمشتق من ال TForm1 .

و TForm1 class فى ال Main.pas تعلن عن اثنين من ال classes  
المضافان لى أولئك الذين ورثتهم فى ال member الأول هو TButton object ،  
ويدعى CloseButton . وال member الثانى من ال TForm1 هو procedure ،  
أو subroutine الذى يؤدي حدوث شئ ما . فى هذه الحالة ، ينهى  
CloseButtonClick procedure البرنامج عندما تضغط زر ال Close الخاص  
بالبرنامج . ذلك هو ال component و event handler الذى قمت بإنشائه بإتباع  
الخطوات الموجودة فى هذا الباب - Delphi يحدد ال code الذى سوف يكتب  
وذلك عند إضافة object على ال form وحسب صفات وال event الخاصة لذلك  
ال object .

## الباب الأول : معلومات حول Delphi 4

ويمكن أن يكون للـ Class ، عناصر توضيحية إضافية بعضها يمكن أن يكون خاص (يستخدم في class unit فقط) أو عام (متاح أيضاً لعبارات في units أخرى) . هذه الحالة التي أمامنا ليس لها أى تعريف سواء أكان public أو private . ويشير التعليقان المحاطان بأقواس الى أين يمكن أن تذهب هذه الأنواع من الـ declarations .

```
private
{ Private declarations }
public
{ Public declarations }
```

إن التعليقات ليس لها تأثيرات في وقت التشغيل . فهي مجرد ملحوظات تصف جانب من البرنامج . وقد يوضح التعليق عبارة محيرة ، أو من الممكن أن يعرف مؤلف البرنامج ، عدد مرات مراجعته ، وحقائق أخرى مختلفة . يمكنك كتابة أى شيء تريده بين الأقواس . ويمكن أن تمتد التعليقات أيضاً لتشغيل أكثر من سطر واحد . وهنا مثال لتعليق متعدد السطور :

```
{ This multiline comment occupies three lines. It
begins with an opening brace on the first line,
and ends with a closing brace. }
```

وبدلاً من الأقواس ، يمكنك استخدام الرموز ثنائية الحرف (\*and\*)- والمتبقية من نظم الـ Pascal عندما كانت كثير من الكلمات الرئيسية والختامية تفتقد علامة الأقواس . على سبيل المثال ، يعتبر هذا تعليقاً مقبولاً للغاية :

```
(*Program author: Your name goes here*)
```

قد تخلط بين أنماط تعليق مختلفة في نفس البرنامج . ولكن اذا بدأت تعليقاً بقوس ، فعليك أن تغلقه بقوس وكذلك ، اذا بدأت تعليق بـ (\* ، فعليك أن تنهيه بـ \*) .

والنوع الثالث من التعليقات- والفريد في الـ Object Pascal ، وتم اقتراضه من ++C- يبدأ بشرطيتين مائلتين ويمتد الى نهاية السطر الجارى . هذا النوع من التعليقات ، لا يمكن أن يطول لأكثر من سطر واحد . على سبيل المثال ، يمكنك استخدام هذا النوع من التعليق لإضافة ملحوظة توضيحية الى تعريف معين .

var

Counter: Integer; // Temporary loop counter

وأخيراً، يوجد فى قطاع interface الخاص للـ unit متغير أو أكثر يسبقه الكلمة الأساسية var. وقد لا يكون للوحدات أى متغير وقد يكون لها واحد أو أكثر. وهذا المثال له متغير واحد.

var

Form1: TForm1;

ويقوم التعريف بإنشاء متغير يدعى Form1 من الـ TForm1 class. بمعنى آخر، يعتبر الـ Form1 object class تحتل مساحة فى الذاكرة. والشكل فى هذا المثال يمثل النافذة الرئيسية لبرنامج Hello، التى تحتوى على event handler يؤدي فعل الزر. مرة أخرى، يكتب Delphi هذا التعريف عندما تقوم بإنشاء الـ form فليس عليك أن تقوم بهذا يدوياً.

ويتشابه الـ class object مع نوع البيانات مثل النوع Integer. يمكنك إنشاء واحد أو أكثر من الـ object لأى class، تماماً كما يمكنك إنشاء متغيرات Integer متعددة.

ويأتى بعد قطاع interface الخاص بالـ unit قطاع الـ implementation حيث يقوم Delphi بإنشاء عبارات لتنفيذ run-time actions، وحيث يمكنك إضافة التعريف وعناصر برمجة أخرى. يمكنك أيضاً كتابة عبارات داخل قطاع الـ implementation على سبيل المثال، قم بتحديد موضع الـ procedure الذى كتبت قبله عبارة الـ Close. ها هى ذى مرة أخرى للإسترجاع:

```
procedure TForm1.CloseButtonClick(Sender: TObject);
begin
    Close;
end;
```

وقبل هذا الـ procedure، تجد هذا الأمر الذى قد يبدو غريباً:

```
{ $R *.DFM }
```

## الباب الأول : معلومات حول 4 Delphi

وبالرغم من أنها تبدو كالتعليق، إلا أن علامة الدولار والواقعة بعد القوس المفتوح مباشرة تنبه Delphi الى أن هذا السطر هو أمر خاص يفتح ويقرأ ملف ينتهى بإمتداد اسم الملف dfm. (إختصار Delphi Form). والنجمة تنبه Delphi الى أن يبحث عن ملف يحمل نفس اسم المشروع، ولكن يتفق بإمتداد اسم الملف dfm. ويحتوى هذا الملف على خصائص ال form التى تم إدخالها مع ال Object Inspector.

وفى آخر سطر من ال Main.pas توجد كلمة النهاية الرئيسية يعقبها نقطة. من الناحية الفنية، تشير كلمة النهاية الرئيسية الى نهاية تركيبة ال unit، وترمز النقطة الى نهاية الملف. ولكن، هذا يعد تمييزاً غير هام. فقط تذكر أن النهاية الأخيرة فى unit ما تنتهى بنقطة.

### ملفات المشروع:

إن خبراء Pascal سوف يتعرفون على ال Hello.dpr على أنه برنامج Pascal [راجع القائمة (١-٣)]. كما هو الحال فى Main unit، يقوم Delphi بإنشاء هذا النص تلقائياً. يعتبر مشروع Delphi بحق مشروع Pascal. إن للمشروع هدفان أساسيان توضيح ال application unit وتنفيذ البرنامج.

إن أول سطر فى Hello.dpr يحدد اسم البرنامج فى تعريف البرنامج - فى هذه الحالة، Hello. فى السطر التالى، إنه يستخدم تعريف يحدد units أخرى يستخدمها البرنامج.

وهذه الحالة التى نحن أمامها تستخدم كلا ال وحدتين. الأولى فى ال unit Forms التى تقدم البرمجة اللازمة لإنشاء Delphi forms فى وقت التشغيل. الثانية هى Main unit فى الملف Main.pas. والسطر الثانى فى قطاع ال uses بالنسبة لل unit ويشير الى أن Main unit تم وضعها فى ملف Main.pas وإن Form1 هو اسم ال form فى هذا ال unit ويكتب Delphi كل هذا ال code تلقائياً:

وأمر ال {\$R \*.RES}. الذى يأتى بعد ذلك، يربط ملف ال binary resource فى الملف ال exe code. وتحتوى ال resource file على ايقونة نظام البرنامج فقط، ولكن يمكن لها ان تحتوى على انواع اخرى من ال resource. على



الرغم من ذلك ، إنك لن تستخدم ملفات resource كثيراً فى Delphi كما تستخدمها فى نظم التطوير الأخرى مثل .C وال C++ .

أخيراً يوجد فى المشروع ثلاث عبارات بين كلمة البداية الرئيسية begin وكلمة النهاية الرئيسية end ، يتبعها نقطة تشير الى نهاية الملف . والعبرة الاولى تبدأ بـ Application object ، والذى يوفر البيانات و code التى تنطبق على التطبيق بأسره . والعبرة الثانية تنشئ الـ form object للتطبيق ، Form1 ، فى الذاكرة والعبرة الثالثة والأخيرة تقوم بتشغيل التطبيق . (ويختلف هذا الـ code فى النسخ الحديثة من Delphi - إذا كنت تستخدم نسخة قديمة من Delphi ، فيمكنك ان ترى عبارتين فقط ، ولكنهما يؤديان نفس الافعال المذكورة هنا) .

**ملحوظة:** يقوم Delphi تلقائياً بإنشاء والحفاظ على الـ source code الملف مشروع الـ Hello.dpr . فنادرأ ما تحتاج الى تعديل عبارات هذا الملف ولهذا السبب ، فإن ابواب هذا الكتاب لا تضع قائمة التالية ملفات مشروعات التطبيقات فى غالبية الحالات ، تقوم بإضافة لغة البرمجة فى unit modules التابعة لـ form معينة وتقوم بادخال هذه البرمجة فى قطاع الـ implementation فقط ، وليس فى قطاع الـ interface .

### امتدادات اسم الملف:

ان تطبيق Delphi النموذجى يحتوى على انواع مختلفة عديدة من الملفات . وينتهى كل اسم ملف بامتداد يعرف محتويات الملف . وتصف القائمة التالية الملفات وامتدادات اسم الملف الخاصة بها التى يقوم Delphi بإنشائها لمشروع Hello الخاص بهذا الباب ، بالإضافة الى مشروعات قليلة اخرى . ولقد شرت ايضاً الى أى من الملفات التى يمكنك حذفها بأمان ، فى حالة اذا كنت تفتقر الى مساحة على القرص ، أو - مثلى أنا - تقوم بالتطبيق وذلك بالتخلص من الملفات الهالكة .

ها هى الملفات التى ينشئها Delphi لمشروع Hello ، مع بعض الملفات الأخرى التى قد تقابلها من حين الى آخره :

● **ملفات ~\*:** والملفات التى بها اسماء تنتهى بـ ~\* . (على سبيل المثال ، Main.~dp) تعتبر نسخ احتياطية من الملفات التى تم تعديلها وحفظها . يمكنك ان

## الباب الأول : معلومات حول Delphi 4

تحذف هذه الملفات فى أى وقت وانت مطمئن ، رغم انك قد تريد الاحتفاظ بها لتعويض الفقد أو التلف فى البرمجة ، أو للعودة الى مراجعة سابقة .

● **ملفات bpg :** ملفات Borland Project Group ، والتي يمكنها تخزين مجموعات من المشروعات تعتبر بحق ملفات نص لأمر ال Make و ملف ال Make يضع قائمة بـ modules التابعة ويوفر الأوامر والمعلومات اللازمة لعملية command-line لتطبيق متعدد ال modules . والمبرمجون دون الخبرة فقط الذين يستخدمون أدوات ال command-line compilation .

● **ملفات dcr :** تحتوى ملفات ال Delphi Component Resource على ايقونة Component كما تظهر على لوحة ال VCL . إنك تستخدم ملفات dcr . فقط عند إقامة Component مخصصة . لا يجب ان تحذف هذه الملفات . اذا فعلت هذا ، فإن Delphi يبدلها بايقونة ثم تخزينها للـ Component عندما تقوم بتركيبها فى ال palette .

● **ملفات dcu :** وتحتوى هذه الملفات على بيانات و code قد إجريت له عملية compiled على سبيل المثال ، ويحتوى Main.dcu على ال codes والبيانات فى source code والتابع لـ Main.pas . يمكنك حذف ملفات ال dcu . الخاصة بالمشروع لان Delphi يعيد إنشائها عندما تقوم بعملية ال compile مرة أخرى .

● **ملفات dfm :** وتحتوى هذه الملفات على binary values تمثل خصائص ال form وكذلك خصائص أى components يتم وضعها على ال form والعلاقات بين ال events والـ event handler يتم أيضاً تخزينها فى ملفات ال dfm . ويقوم Delphi بنسخ هذه المعلومات فى ملف ال exe . لا تقم ابداً بحذف ملفات ال dfm . وعليك معاملتها بنفس الحرص الذى تعامل به ملفات ال source code .

● **ملفات dll :** وتحتوى هذه الملفات على ال code الخاص بـ dynamic link libraries . بالرغم من انه بإمكانك حذف ملفات ال dll . وانت مطمئن حيث انه لديك ال source code لا تحذف ملف ال dll . الا اذا كتبتة أو كنت واثقاً من مصدره .

● **ملفات dof :** وتقوم هذه الملفات بتخزين كافة عناصر المشروع . ويعتبر ال dof هو إسم الإمتداد الخاص بـ Delphi Options File اذا كنت تستخدم أمر ال

**ProjectOptions** لتغيير العناصر متنوعة لهذا المشروع، فإن Delphi يحفظ مواضيعك فى ملف `dof`. الخاص بالمشروع. والسبب الوحيد لحذف هذه الملفات هو للعودة الى العناصر الأساسية للمشروع. ويحل هذا النوع من الملفات محل ملف `opt`. فى النسخ السابقة من Delphi.

● **ملفات dpk**: هذا النوع من الملفات يحتوى على `package` والتي تكون فى الغالب عبارة عن مجموعات من `units`. على سبيل المثال، ان `VCL` بأكمله يتوفر فى `package`. لا تحذف هذه الملفات ابداً. ان `package` فى الحقيقة عبارة عن ملف `Pascal source code` - فهو يضم `code` أجرى له عملية `compiled` والنتائج `files (dcu)`. ولكنه لا يضم هو فى ذاته أى `code`.

● **ملفات dpr**: وهو اختصار لـ `Delphi Project`. وتعتبر ملفات `dpr`. فى الحقيقة ملفات `Pascal source code` فى معظم الحالات، يجب الا تقوم بتعديل ملفات `dpr`، رغم ان واضعى البرامج الخبراء فى `Pascal` قد يفعلون هذا بحذر شديد. يقوم Delphi بإنشاء ملف `dpr`. عندما تقوم انت لأول مرة بحفظ مشروع تطبيق جديد. لا تحذف ابداً ملفات `dpr`.

● **ملفات dsk**: تقوم هذه الملفات بتخزين مواصفات `application's desktop` ولكن هذا فقط يحدث عند اختيارك لـ `Autosave` مع إختيارك للاختيار `ToolsEnvironment`. يمكنك حذف ملفات `dsk`. اذا لم تكن تريد الاحتفاظ بترتيب نوافذ `Delphi's`، أو اذا اردت العودة الى التخطيط الافتراضى حسب نظام البرنامج. رغم ذلك، تقوم ملفات `dsk`. بتخزين اسماء مسارات المشروع، حتى اذا ما حركت مشروعك الى مواضع أخرى، أو اذا ما قمت بتخزين `modules` فى دلائل متعددة، فيجب الا تحذف ملفات `dsk`.

● **ملفات exe**: كما تعرف، ان الملفات المنتهية بـ `exe`. هى ملفات يطلق عليها `executable code` وكما ذكرت، يمكن ان يقوم Delphi بإنشاء `exe code`. `file`. وهو الوحيد الذى يجب عليك ان توزعه الى مستخدمى تطبيقك يمكنك باطمئنان حذف ملف `exe`. للمشروع لأن Delphi يعيد إنشاءه عندما تجرى عملية `compile`.

## الباب الأول : معلومات حول Delphi 4

● **ملفات opt.** :راجع ملفات الـ .dof. ، والتي تحمل محل ملفات opt. فى النسخ الحديثة من Delphi . ان ملف الـ opt. قد اختفى الآن . اذا وجدت واحداً - عند تحديث تطبيق قديم ، مثلاً - يمكنك حذفه وانت مطمئن بعد فتح وحفظ المشروع مستخدماً نسخة احدث من Delphi .

● **ملفات pas.** :تحتوى هذه الملفات على الـ Pascal source code . وتطبيق Delphi النموذجى له ملف pas. واحد لكل unit تابعة لـ form معينة ، رغم ان مطورى Pascal الخبراء قد يقومون بتخزين برمجة أخرى فى ملفات الـ pas . يمكنك عرض ملفات الـ pas. باستخدام الـ Delphi's code editor ، أو بأى ASCII text editor . لا تحذف ملفات الـ pas .

● **ملفات res.** :الملفات ذات اسماء تنتهى بإمتداد الـ res. وتحتوى على binary resources مثل ايقونة نظام البرنامج وأى bitmaps أخرى ، وتبدو ملفات الـ Resource تشابه بشكل واضح مع برمجة الـ nonvisual Windows من خلال الـ Borland Pascal ، CC ، أو C++ عنها فى الـ Delphi . فى الـ Delphi ، استخدم الـ Image Editor فى قائمة الـ Tools لإنشاء وتعديل ملفات الـ res . لا تقم ابدأ بأية تغييرات لملف الـ resource للمشروع ، والذي له نفس اسم المشروع ولكن ينتهى بامتداد اسم الملف الـ res. اذا فعلت هذا عندما يقوم Delphi بإعادة انتاج هذا الملف ، يتم فقد أى resource تكون قد أضفته إلى الأبد .

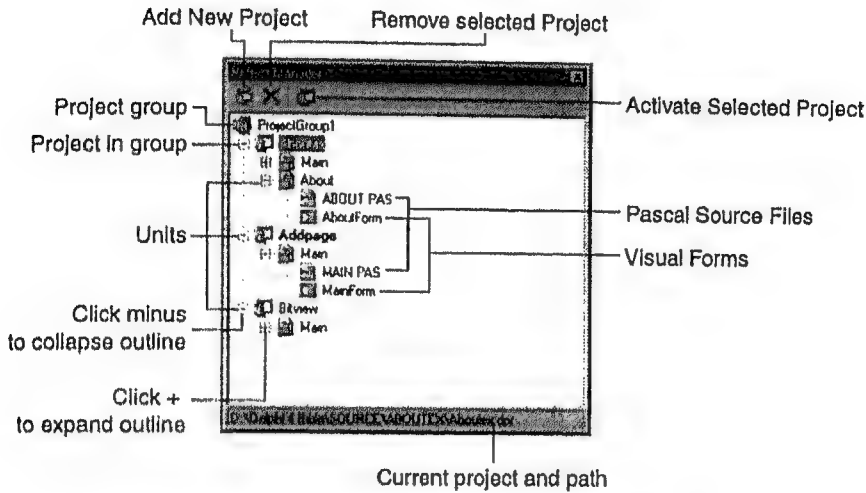
**تحذير:** لا تحذف ابدأ ملفات لها اسماء تنتهى بـ dfm أو dpr . أو pas. ، وتحتوى هذه الملفات على source code وخصائص التطبيق . وعند تدعيم تطبيق اثناء التطوير ، فإن هذه هى الملفات الهامة التى يجب ان تحفظها .

## :Project Manager

الـ Project Manager فى Delphi 4 يعتبر اداة جديدة تماماً تحمل محل النسخة السابقة ، والذي اجرى له تغيير بسيط من النسخة الأولى Version 1 والـ Project Manager لإنشاء ملفات Make لإجراء عملية compile من خلال command line باستخدام الـ Make utility . ويستطيع الـ Project

Manager الجديد ان يجمع مشروعات أو أكثر مرتبطين، والذي يمكن فيما بعد إجراء عملية ال compile لهما بأمر واحد.

يوضح الشكل (٥-١) نافذة ال Project Manager الجديد. يظهر الشكل ثلاث من المشروعات الموجودة على القرص المدمج الخاص بهذا الكتاب والتي، قمت أنا بتصنيفها في مجموعة مشروع واحدة.



شكل (٥-١)، نافذة ال Project Manager الجديد فى 4 Delphi

### استخدام ال Project Manager :

تعتبر غالبية أوامر ال Project Manager بديهية. اختر مشروعاً واضغط Activate Selected Project speed button يمكنك ان تضيف مشروعاً خالياً وجديداً بضغط Add New Project speed button. أو يمكنك حذف مشروع بضغط زر ال Delete Project. يمكنك أيضاً اضافة وحذف ملفات فردية الى مشروع أو آخر.

تتغير اوامر ال pop-up menu الخاصة بال Project Manager اعتماداً على أى نوع من العناصر الذى تم إختياره على سبيل المثال، قم بإختيار اسم مشروع واضغط زر الفأرة الأيمن للحصول على قائمة بالأوامر التى يمكنك تطبيقها على البرنامج. قم بإختيار اسم ملف واضغط زر الفأرة الأيمن للحصول على قائمة بالأوامر التى تنطبق على ملفات فردية.

## الباب الأول : معلومات حول Delphi 4

وكثير من اوامر الProject Manager تتوفر ايضاً في قائمة الProject . وهذه الاوامر تتغير لتعكس المشروع الذى تم اختياره حالياً ، والذي يتم عرضه بخط سميك . يمكنك فتح وإغلاق اية ملفات فى أى مشروع معروض فى نافذة الProject Manager ولا يجب ان تكون الملفات متعلقة ببعضها .

على سبيل المثال ، باستخدام المعلومات الموجودة فى الفقرة التالية حول إنشاء مجموعات مشروع ، يمكنك فتح الsource code unit فى مشروعين منفصلين ، لا علاقة لهما ببعض . ويعتبر هذا مريحاً للعمل على مشروعات متعددة والتى ، قديماً ، كان يجب عليك ان تقوم بفتحها وإغلاقها منفردة (أو تحميلها فى نسخ متعددة من Delphi) .

**نقطة هكتر:** يمكنك توصيل نافذة الProject Manager لجعلها اسهل فى العثور عليها . اضغط واسحب حد النافذة الى اليسار أو اليمين من الcode editor ، ثم أترك زر الفأرة . إفصل النافذة بالضغط وسحب حدها الأعلى ثم اترك زر الفأرة .

### إنشاء project groups

يستطيع الProject Manager الجديد ان يجمع مشروعات متعددة ، والذي قد يكون مفيداً لإجراء عملية الcompile لأكثر من مشروع متصل ببعضه فى وقت واحد . اتبع هذه الخطوات لإنشاء sample project group والتى استخدمتها لإعداد الصورة الموجودة فى شكل (١-٥) :

١- اختر File|New واختر New page tabs . اضغط مرتين ايقونة الProject Group لإنشاء ملف Project Group جديد وافتح نافذة الProject Manager .

٢- لإعطاء project group اسم مختلف (والاسم حسب نظام البديل الافتراضى للبرنامج هو ProjectGroup1) ، قم بحفظ المشروع فى أى دليل . اذا كنت تجمع مشروعات موجودة فى مجموعة ، فقد تريد إنشاء دليل منفصل للملف المشروع ، ولكن هذا يرجع لك . يمكنك تخزين المجموعة فى أى مكان تريده . لحفظ الProject Group اخترها فى نافذة الProject Manager ، واضغط زر الفأرة الأيمن فى النافذة . اختر ..... Select Save Project وأعط اسم ملف .

٣- افتح pop-up menu مرة أخرى (تأكد من ان بند ال ProjectGroup1 قد تم اختياره). واختر أمر ال Add Existing Project. تصفح أى دليل تطبيق لـ Delphi، وافتح ملف مشروع dpr. لإضافته الى المجموعة. يمكنك أيضاً استخدام امر ال Add New Project لإنشاء مشروع جديد للمجموعة.

٤- احفظ المجموعة التى تم تعديلها. ولاستعراض ال Source code الخاص بال project group، افتح مرة أخرى ال pop-up menu واختر ال View Project Group Source. توضح القائمة (١-٤) ال source code لل project group للمشروعات الموضحة فى شكل (١-٥). (هذه القائمة غير موجودة على القرص المدمج - فهى لا تخدم اية اهداف عملية وموضحة هنا للعرض فقط). كما ترى، يعتبر ملف ال project group فى الحقيقية ملف نص Make، والذي يمكن استخدامه مع ال Make utility لإجراء عملية compile لأكثر من مشروع باستخدام ال Delphi command-line compiler.

القائمة (١-٤): ملف project group هو فى  
الحقيقية عبارة عن command-line Make file

```
#-----
VERSION = BWS.01
#-----
!ifndef ROOT
ROOT = $(MAKEDIR)\.
!endif
#-----
MAKE = $(ROOT)\bin\make.exe -$(MAKEFLAGS) -f$**
DCC = $(ROOT)\bin\dcc32.exe $**
BRCC = $(ROOT)\bin\brcc32.exe $**
#-----
PROJECTS = Aboutex Addpage Bitview
#-----
```

## الباب الأول : معلومات حول 4 Delphi

default: \$(PROJECTS)

#-----

Aboutex: ..\Delphi 4

Bible\SOURCE\ABOUTEX>Aboutex.dpr

\$(DCC)

Addpage: ..\Delphi 4

Bible\SOURCE\ADDPAGE/Addpage.dpr

\$(DCC)

Bitview: ..\Delphi 4

Bible\SOURCE\BITVIEW/Bitview.dpr

\$(DCC)

ولن اخوض في الاوامر الموجودة في ملف ال project group - فإن المطورين ذوى الخبرة فقط هم الذين يريدون فهمها . على الرغم من ذلك ، فإن اهمية المجموعة هي انها تستطيع تبسيط عملية تصفح إجراءات عملية ال compile لأكثر من مشروع وهذه الخاصية لل Project Manager الجديد تجعله مثيراً للفريق من المبرمجين الذين يعملون في إعداد أكثر من مشروع . ويمكن لمجموعة واحدة ان توفر الوصول الى كل ال source code الخاص للتطبيق من داخل دورة واحدة لـ Delphi .

### افكار للمستخدم الخبير

● في نسخ Delphi القديمة وبخاصة اذا لازالت تستخدم Windows 3.1 ، اذا تلقيت رسالة خطأ تقول "out of resources" ، خلال تركيب Delphi ، فإنه غالباً يكون أحد أمرين إما out-of-date ، أو nonstandard video driver . استخدم ال Windows Setup لتغيير ال driver الى Standard VGA . يمكنك هذا من تركيب ال Delphi . اتصل ببائع ال VGA الذى تتعامل معه لتحصل على driver حديث . فلا يجب ان يحدث هذا مع النسخ الاحدث لـ Delphi .



• يصعب عليك التعامل مع ال form و editor windows حيث أنهما غالباً ما يخفيان بعضهما البعض ويصعب إيجادهما . وللانتقال بينهما بطريقة اسهل ، اضغط F12 و F11 . جرب هذا الآن لترى ما تفعله هذه المفاتيح .

• ان hint boxes الخاصة بـ Delphi ، والتي تظهر عندما تضع المؤشر على ايقونة ال component أو ال speed button التابعة لل form التابعة له . ولا تظهر نفس ال Hint boxes اذا كنت تتعامل مع تطبيق آخر .

على سبيل المثال ، اذا قمت بالتعامل مع Start menu bar فى ال Windows 95 ، فإن Delphi hint boxes تختفى فإذا لن ترى hint boxes بعد الانتظار لمدة معقولة ، اضغط مؤشر الفأرة على Delphi's title bar أو على أى نافذة أخرى مثل ال Object Inspector ، ثم حاول مرة أخرى .

• للحصول على تقرير مفصل اثناء عملية ال compile وال link اختر أمر ال ToolsEnvironment Options..... ، اختر Preferences page tab وقم بعرض التقرير المفصل . عندما تقوم بعد ذلك بعمل compile ، link لبرنامج ما يعرض ال Delphi ، تقرير للحالة والذي يوضح الملف وعدد السطور التى حدث لها عملية compile ، link وهذه المعلومة لا تعد ذات قيمة للغالبية المطورين ، الا انهاء تؤكد ان شيئاً ما يحدث خلال عملية compile . كن على حذر : ان عرض displays يستقطع وقتاً من عملية compilation ، واذا قمت بكتابة كثير من التطبيقات الصغيرة ، فقد تأخذ وقتاً اطول لعرض ال dialog بالنسبة إلى الوقت اللازم لإجراء عملية compile لهذه البرامج .

• اضغط مرتين على ال form أو code editor ليزداد حجمها حتى يتناسبان مع حجم الشاشة . واذا كان هذا يخفى ازرار وقوائم Delphi ، اضغط Alt لتجعل toolbars و Delphi's component palette تظهر مرة أخرى يمكنك أيضاً أن تفعل هذا بضغط Alt+F +Alt- على سبيل المثال - لفتح قائمة ال File . اضغط Alt ، اذا لزم الأمر ، للعودة الى expanded window وطبيعياً يظهر ال Delphi ال expanded editor تحت VCL palette ، لزيادة حجم نافذة لتغطى كل الشاشة ، اختر أمر ال ToolsEnvironment Options ، واختر Display display page tab (سابقاً Editor ) ، واغلق ال Zoom الى مساحة شاشة الكاملة .

## الباب الأول : معلومات حول Delphi 4

● يمكنك استخدام الـ editor window لعرض أى Pascal source code file (والذى ينتهى غالباً بامتداد اسم الملف .pas) أو ملف نص (غالباً ما ينتهى بـ .txt). اختر أمر الـ File|Open.... أو اضغط Open-file speed button . وللإستفادة من إبراز الجمل الخاص بـ Delphi ، يجب ان ينتهى الملف بامتداد اسم الملف الذى تم ادخاله فى مربع نص الـ Tools|Environment Options..... Editor Syntax-extensions . وكل امتداد يكون مفصول بفصلة منقوطة .

● وطبقاً للعرف ، ان العبارات الواقعة بين كلمة البداية الرئيسية وكلمة النهاية الرئيسية - وتسمى من الناحية الفنية block - تحدها مسافتان [راجع القائمة (١-١) و (٢-١)].

وترك المسافات ليس مطلوباً ، ولكنه يساعد على جعل البرنامج مفهوماً بتوضيح أى العبارات تتبع بعضها . وطبقاً للعرف ، ان العبارات التى تحدها مسافات لها نفس المستوى تكون مرتبطة نوعاً ما .

● من خلال الـ display ٦٤٠×٤٨٠ ، اختر Tools|Environment|Options Display page tab (سابقاً Editor) . حدد خط النص بـ Courier New (والبنط الأصلى) وحدد حجم بـ ٩ نقاط . وبهذا الوضع ، والذى يعتبر مفيداً للـ laptops ، يُعرض غالبية الـ source code فى الـ editor الذى يحدد حجمها حسب النظام الافتراض للبرنامج دون ان يتطلب هذا ظهور الـ horizontal scrolling . اذا كانت عينيك افضل من عيناى ، جرب حجم الثمان نقاط لترى مزيداً من النص . قد تريد ايضاً ان تجرب مع خيار الـ mappings الخاص بـ dialog's Keystroke ، والذى يشكل أوامر لوحة مفاتيح الـ editor . اختر الـ Default اذا كنت مستخدم خبير فى الـ Windows ، واذا كنت على علم بـ Borland Pascal أو الـ C++ ، أو Brief أو Epsilon .

● لإختبار الـ multiple running applications ، ابدأ من خلال الـ first instance فى الـ Delphi ثم استخدم الـ Windows Explorer لبدء الثانى . أو ، لإختيار الـ instance الثانية أو instances المتعاقبة ، ابدأ الـ instances الأولى مع الـ Explorer ثم قم بتشغيل الـ instance النهائية من Delphi . استخدام اوامر debugging فى قائمة الـ Run لإدخال عبارات البرمجة للـ instance التى بدأت مع Delphi .

• ان الخصائص وال event relationships يتم تخزينها فى ملفات dfm. وهى ليست ملفات نص - إنها binary data file - ويجب الا تستخدم text editor لتعديلها. على الرغم من ذلك، اذا قمت بفتح ملف file. باستخدام File|Open.....، يقوم Delphi بعرضه كانه نص. (جرب هذا الآن بفتح ملف Hello.dfm الخاص بمشروع Hello) وسوف يتعرف واضعى برامج Pascal على هذا النص على انه object-type declaration والتي تحدد خصائص وعلاقات event لل component. إنها ليست Pascal حقيقية، ولكن قريبة جداً ولكن فريدة فى ال Object Pascal. ولحفظ نسخة قابلة للطبع من هذه المعلومة، اختر File|Save As..... واحد تسمية الملف بامتداد اسم ملف text.. يمكنك ايضاً ان تفعل العكس - افتح ملف النص، قم بتعديله، واحفظه مع امتداد اسم الملف dfm. لتحويل النص مرة اخرى الى binary. والمبرمجون الخبراء قد يفعلون هذا، على سبيل المثال، لتعديل خصائص المشروع، رغم انه من الاسهل، فى غالبية الحالات، استخدام نافذة ال Object Inspector لهذا الغرض.

### المشروعات التى يمكنك تجربتها

(١-١): افتح مشروع Hello واختر زر Close لل form. قم بتغيير خاصية ال Default لهذا الزر لتصبح True اضغط F9 لإجراء عملية compile، link وتشغيل البرنامج الذى تم تعديله. لانتهاء البرنامج، يمكنك الآن ضغط Enter، والذى أختير على أنه ال control الأساسى حسب النظام فى ال Window. وعودة إلى ال Delphi، حدد خاصية Cancel فى زر ال Close ب True. الآن يمكنك تشغيل البرنامج، يمكنك ايضاً ضغط Esc للإنتهاء. ولكن غالباً ما يكون component واحد فقط فى ال form يكون له خاصية Cancel أو Default الخاصة به محددة ب True.

(٢-١): حاول إضافة components اخرى الى Hello أو إنشاء تطبيق اختياري جديد. ان بعض الدقائق تقضيها فى اللعب قد تساعدك على ان تصبح اكثر اعتياداً على تخطيط Delphi.

## الباب الأول : معلومات حول Delphi 4

(٣-١): اختر أمر ال Tools|Environment Options لتنظيم بيئة

Delphi. استخدم Editor and Colors page tab لتعديل

خصائص ال editor، ولأختيار ألوان إبراز تركيب الجمل.

يستطيع Delphi عرض الكلمات الرئيسية وعناصر أخرى في

ملفات source code بنص مائل، وألوان مختلفة، وخط سميك

وما إلى ذلك. إن إبراز تركيبات الجمل ليس للعرض فقط- إن

الأختيار الصائب للألوان وخصائص النص يساعد على جعل

البرامج أكثر وضوحاً بإبراز العبارات والكلمات الرئيسية

والتعليقات والعناصر الأخرى.

(٤-١): ولزید من الاطمئنان، اختر أمر ال Tools|Environment

Options، اضغط Preferences tab، وقم بتشغيل خيارى ال

Autosave. والخيار الأول، ملفات ال Editor، يقوم تلقائياً

بحفظ كل الملفات فى مشروع عندما تقوم بعملية ال compile و

ومن ثم تنفيذه، والخيار الثانى، Desktop، يحافظ على ترتيبات

نوافذ Delphi. مع هذا الخيار، وعند ماتعيد بدء Delphi، يمكنك

أختيار ما سبق فى الدورة السابقة.

**Tip** **فكرة:** إذا كنت تقوم بتشغيل كثير من البرامج التجريبية القصيرة لأختيار

خصائص ال Delphi، فقد تريد إيقاف خيار ملفات Autosave

Editor لخفض نشاط القرص. وعند تطوير تطبيق كبير ذو ملفات عديدة

مختلفة، فقد يمنع هذا الخيار الفقد الذى يحدث للـ ham بطريق الصدفة.

### ملخص:

● يعتبر ال Delphi نظام تطوير تطبيقات سريع، مناسب لإنشاء عينات

نموجية من ال Windows و تطبيقات تامة تضاهى وتفوق فى السرعة والكفاءة

برامج كتبت فى ال C، C++، Borland Pascal 7.0، و Visual Basic،

وكذا البرامج التى تم إنشاءها بوسائل أخرى إن الخصائص الجديدة فى ال Delphi 4

تشمل امتدادات لغة ال Object Pascal، و Project Manager جديد، و

Module Explorer لملفات ال source code للـ unit. ومن الجديد فى

Delphi أيضاً خصائص ال debugging وتطور ال VCL و client-server component وقاعدة البيانات .

• ويتكون تطبيق Delphi من واحدة أو أكثر من ال form والتي تضع عليها ال VCL components ويمكنك تعديل خصائص ال form و components مثل ال Names و Captions الخاصة بهم ، ويمكنك انشاء Pascal procedures لإداء ال events مثل استخدامك لأحد ازرار الفأرة .

• وتوفر ال Code insights قوائم لل parameters وال code completion ويمكن ان تساعدك فى توفير وقت الكتابة والوقت المنقضى فى البحث فى ال online help screens و documentation وبالإضافة الى خصائص التحرير المفيدة هذه ، هذا بالإضافة الى ميزة ال tool-tip expression evaluation والذى يظهر قيمة المتغير فى حالة ال debug .

• عندما تقوم بتشغيل تطبيق ، يقوم Delphi بعملية ال compile و link لل source code و modules الأخرى لإنشاء ملف .exe . وهذا هو الملف الوحيد الذى تحتاج ان توزعه على مستخدمى برنامجك .

• يمكن لل Project Manager الجديد الخاص بـ Delphi ان ينشئ project group لمشروعات متعددة وهذا يبسط عملية المتصفح فى ملفات مشروعات متعددة ، وينشئ أيضاً ملف Make لإجراء عملية ال compiling لمشروعات متعددة . وتمكنك خاصية ال Project Manager الجديدة من تصفح أجزاء للcode .

فى الباب التالى والمقدمة لـ Visual Components ، سوف تعرف المزيد عن كيفية إضافة ال component فى ال application's form ويقدم الباب مجموعة ال components فى ال Delphi مع أمثلة من التطبيقات التى تظهر كثير من الخصائص وال event الخاصة بال Visual Component .

\*\*\*\*\*

# الباب الثانى

## معلومات حول

### ال Visual Components

#### محتويات الباب:

Visual Component Library •

Standard components •

Additional and Dialogs components •

System components •

Win32 components •

إن البرمجة من خلال ال visual components يعد أمر غاية فى السهولة .  
يمكنك أن تنجز الكثير بأقل مجهود .

وال visual component ، هو object مثل أى زر تقوم بإضافته على ال form .  
فى الواقع ، إن ال form هى أيضاً object تستطيع أن تحمل objects أخرى . وكما تطور  
تطبيقات ال Delphi ، فسوف تقضى معظم وقتك فى إضافة ، وتعديل ال visual  
component objects ، لذا فمن المهم أن تفهمهما بتركيز . وال Visual components  
هى العناصر الأساسية التى تركز عليها أساسيات ال code الخاص بك .

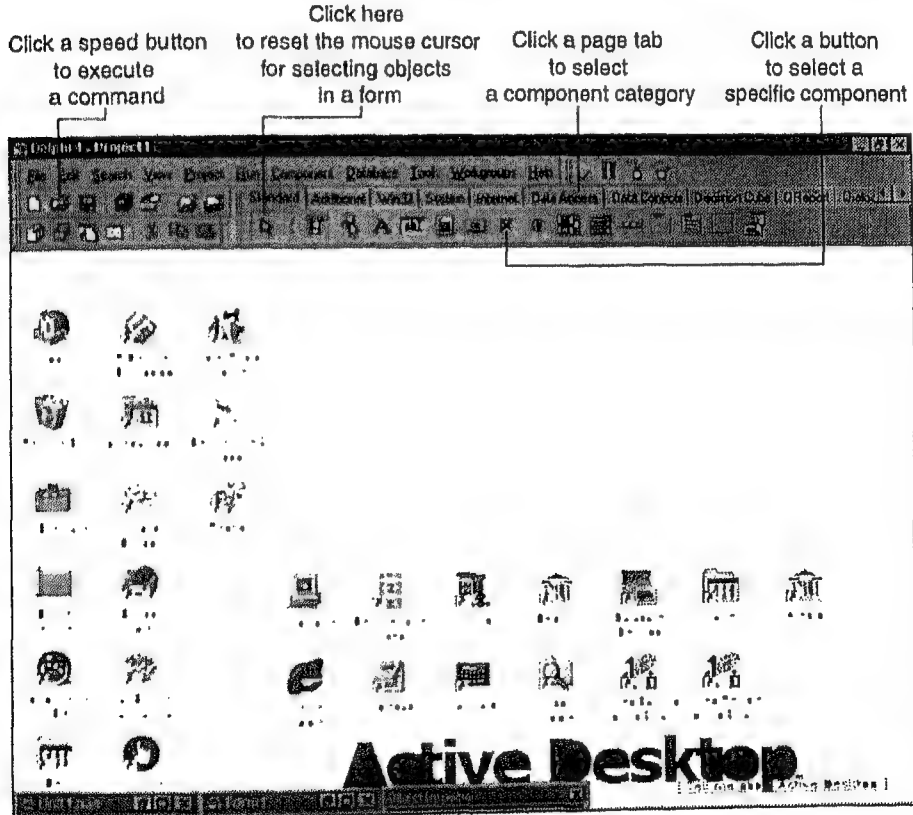
#### ال Visual Component Library :

مع ال Visual Component Library (VCL) ، يمكنك إنشاء تطبيقات  
مفيدة مع جزء يسير من البرمجة . ولكن لا تأخذ كلامى ثقة ؛ جرب أمثلة من  
التطبيقات الموجودة فى هذا الباب لعرض التطور السريع للتطبيق مع Delphi .

بعد التمكن من أساسيات ال visual components فى هذا الباب ، يمكنك أن تجد مزيداً من المعلومات عن component معينة فى أبواب الجزئين الثانى والثالث .

## :Visual component categories

لإختيار component ، اختر page tab مثل Dialogs فى ال Delphi's component palette [أنظر شكل (١-٢)]. وفيما يلى قائمة بعناصر ال Delphi visual component فى ترتيب ظهورها على الشاشة . إن عدد وترتيب الفئات على شاشتك قد يختلف اعتماداً على نسخة وطبعة ال Delphi التى معك ، وخيارات التركيب ، ومكونات إضافية تكون قد ركبته . والترتيب قد يختلف أيضاً إذا كنت قد استخدمت ال Tools/Environment Options لترتيب أيقونات ال palette .



شكل (١-٢): لوحة ال visual component فى ال Delphi وتظهر ال page tab التى تنظم ال component فى فئات

## الباب الثاني : معلومات حول ال Visual Components

● **Standard**: Standard Windows، مثل الأزرار، و labels، ومربعات الإدخال، والقوائم، و check boxes و scroll bars.

● **Additional**: Custom controls مثل bit buttons الذى لديها خاصية graphic images والتى تسمى (glyphs)، toolbar speed buttons، string graphical grids، scroll boxes، graphics shapes و bitmap، static text، check box lists، window splitters، و Simple images، و chart tool.

● **Win32**: NT controls و Windows 95 ذات ال ٣٢ بت القياسية بما فى ذلك track and، rich-text editors، image lists، page controls، status bars و date-time picker، animation windows، progress bars و tree list view objects، and toolbars.

● **System**: System service components وتشمل، paint boxes، و OLE و Dynamic Data Exchange، timers، و multimedia player.

● **Internet**: Internet components و ال World Wide Web وأدوات connection لـ Internet/Intranet protocol، FTP، HTML، HTTP، و UDP، TCP، و objects.

● **Data Access**: ال Components المخصصة للتعامل مع قواعد البيانات وذلك من خلال tables، SQL queries، وبالإضافة أنك تستطيع مع هذه الأشياء إنشاء تقرير بواسطة QReport.

● **Data Controls**: Data-aware components للتعامل مع البيانات الناتجة عن قواعد البيانات. فاستخدام هذه ال components والتى تشمل edit boxes، memos، check boxes، pictures، و lists.

● **Decision Cube**: هذه ال Components، تكون تقرير يطلق عليها decision reports وهى على سبيل المثال cross tabulations مع المعلومات الخاصة بقواعد البيانات- ولا توجد إلا فى نسخة ال client-server.



• **QReport**: مجموعة كبيرة من ال objects متصلة بـ data-aware component . وهذه ال component تحمل محل تطبيق ReportSmith الذى كان متوفراً فى النسخ الأولى من Delphi . لمزيد من المعلومات حول QReport component ، أنظر "Additional and Dialogs Components" فى هذا الباب وكذا الباب السابع عشر . أضيف الى ذلك أن Delphi يقدم user manual من خلال Microsoft Word لهذه component الكثيرة . أنظر ملف Qrpt2man.doc الموجود فى دليل Quickrpt الخاص بـ Delphi .

• **Dialogs**: مخصصة هذه ال component لأداء مهام معينة مثل إختيار الملفات والأدلة ، إختيار الخطوط والألوان ، والطباعة وإيجاد أو إحلال بيانات فى الوثائق .

• **Win3.1** : وهذه ال component تتطابق مع ال components التى كانت فى نسخة Delphi الأولى التى تعمل فى بيئة Windows 3.1 . وجميع ال components فى هذه الفئة لها بدائل ذات ٣٢ بت . ولا يحتاج غالبية مبرمجى Delphi الى هذه ال components ، وهذه الفئة يمكن حذفها من لوحة المكونات اذا أردت . لعمل هذا ، استخدم ال ..... Tools\Environment Options واضغط . Palette page tab.

• **Samples** : هناك components قد تم وضع ال source code الخاص بها فى Source\Samples الخاص بـ Delphi . والغرض من هذه ال components هو فى الغالب ان تكون مرشدة لك فى إنشاء ال component الخاصة بك ، وتجدها مفيدة لك فى التطبيقات ولم تتغير هذه الأمثلة كثيراً عبر تاريخ Delphi ، ويمكن حذف هذه الفئة وانت مطمئن فهناك component تعطى نفس الإمكانيات .

• **ActiveX** : تمثل الجزء الثالث من ال sample components وهى توضح استخدام وبرمجة ال ActiveX بإستخدام Delphi . لمزيد من المعلومات حول ActiveX ، انظر الباب العشرين من هذا الكتاب ، "بناء ال Custom Components" .

## الباب الثانى : معلومات حول الـ Visual Components

• **MIDAS** : وهى مثل الـ ClientDataSet, RemoteServer, Provider وغيرها لإنشاء تطبيقات الـ multitiered database وتتوفر هذه الخاصية فى النسخة client-server.

### الأمثلة التطبيقية:

تقوم ثلاث أمثلة تطبيقية فى هذا الباب بأداء مهام نافعة أثناء عرض الاساليب التى تحتاجها فى برمجة Delphi. يظهر تطبيق الـ MemoPad التعامل مع الملف والقائمة. ويظهر تطبيق الـ BitView كيفية استخدام graphics and dialog boxes. ويظهر تطبيق الـ DClock كيفية استخدام Timer component لإنشاء عمليات تعمل أثناء تنفيذ عمليات أخرى.

إننى اقترح عليك إنشاء هذه التطبيقات بنفسك باتباع الإرشادات والمقترحات الموجودة فى هذا الباب - فهى طريقة عظيمة لتعليم تقنيات Delphi. لا تقم بتشغيلها - فلن تتعلم أى شئ بهذه الطريقة. بالطبع، ان القرص المدمج الخاص بهذا الكتاب يوفر جميع التطبيقات وملفات الـ source code الخاصة بها. انسخ دلائل المشروع المختارة من على القرص المدمج الى محرك القرص الصلب الخاص بك، ثم استخدم الـ FileOpen.... لتحميل ملف dpr. الخاص بالمشروع. اضغط F9 لتنفيذ التطبيق. يمكنك ايضاً استخدام الملفات المتوفرة اذا تعثرت أثناء إنشاء البرامج الخاصة بك. انظر فقرتى "Win32 Components" و "Additional and Dialogs Components" فى هذا الباب للتعرف على بعض الـ component الأكثر تعقيداً فى Delphi.

**على القرص:** جدول (٢-١) يضع قائمة بالأمثلة التطبيقية هذا الباب، والتى تم تخزينها على القرص المدمج المرفق بهذا الكتاب فى دلائل تتفق مع اسماء التطبيقات. على سبيل المثال، سوف تجد ملفات تطبيق الـ MemoPad فى دليل الـ MemoPad. ويوضح الجدول ايضاً فئات الـ components التى يستخدمها كل تطبيق.



### : Standard Components

مع الـ Standard component، يمكنك إنشاء Windows objects وعناصر اساسية عامة مثل الازرار و check boxes والقوائم. لان الـ Standard

components سهولة الاستخدام نسبياً، فإنها تعتبر مقدمة جيدة لبرمجة ال visual component .

جدول (١-٢)، أمثلة تطبيقية عن visual component

التطبيق	Category	البيان
MemoPad	Standard	Memo pad text file editor . يظهر كيفية برمجة أوامر القائمة الرئيسية pop-upmenu وكيفية قراءة وكتابة ال text files .
BitView	Standard ، Additopnal Dialogs	Bitmap file viewer . يفتح ويعرض أى Windows bitmap أو أى أيقونة أو ملف كبير أو أى ملف للرسومات آخر . ويعرض لك كيفية البرمجة باستخدام graphical and dialog component .
Dclock	Standard ، System	ساعة رقمية . يوضح كيفية برمجة ال timer component الذى يحدث فى نفس الوقت مع مهام أخرى .

### تصميم التطبيقات باستخدام ال component objects:

إبدء تنفيذ برنامج Delphi اذا لم يكن قد بدأت بعد، اذا كان تطبيقاً قد تم تحميله بالفعل، استخدم File/New Application لإضافة form جديدة للتطبيق، اضغط Standard page tab تحت ال component palette لعرض ايقونات ال component الموجودة فى هذه الفئة [راجع الشكل (١-٢)] .

إختر ايقونة ال component على سبيل المثال، اضغط الزر الذى يحمل حرف ال A، والذى يمثل Label component حرك المؤشر داخل نافذة ال form اضغط زر الفأرة الأيمن وأضف ال Label على ال form، فى الموقع الذى تريده ولا تقم بعملية ال drag لل component على form وبدلاً من ذلك أختر ال component icon وحرك المؤشر الى ال form وأضغظ بزر الفأرة وأضف ال object على ال form أضغظ مرتين متتاليتين على الفأرة أو على أى component

## الباب الثاني : معلومات حول الـ Visual Components

لإضافته على الـ form. بعد ذلك، اضغط واسحب الـ component لتحريكه. لإضافة نسخ متعددة من نفس نوع component، إضف واحد على الـ form، وتاركاً إياها مختاره واضغط **Ctrl+C** يتبعها **Ctrl+V** لكل نسخة جديدة.

لحذف الـ component، إختره واضغط **Delete** كن في غاية الحذر وانت تفعل هذا - يمكنك ان تلغى آخر. اذا قمت بحذف الـ component، فإن الثاني فقط يمكن استعادته. لإسترجاع الـ object الذي قد تم حذفه اضغط **Edit/Undo** (أو اضغط **Ctrl+Z**).

**ملحوظة:** لتفادي الغموض الذي قد يحدث في هذا الكتاب، فإن مصطلح الـ component يعود على ايقونة في **VCL palette**؛ **component object**، (أو **object** فقط) هي عبارة **component** تم إضافته على الـ form. اذا صادفتك إحدى التعليمات في هذا الكتاب تقول: اضغط **object** أو **component object** إختار من على الـ form كما وأن اسماء الـ Class تبدأ بحرف الـ **T**؛ لذا فإن برمجة **Label component** توجد في **TLabel class**. والـ **Object** قد تم تسميتها باستخدام اسماء الـ **component** وتعد **Label1**، **Label2**، **FileLabel** امثلة للاسماء الجيدة لانها تذكر بأنها **Label component object**.

إضف **component object** قليلة على الـ form وذلك باختيارها من الـ **Standard palette** واضغط الفأرة على الـ form بعد ان تضيف الـ **object** (لا يهم العدد بالضبط والنوع)، قم بتجربة التجارب التالية لتعتاد على اوامر ترتيب الـ **component**:

- لتحريك وإعادة تحديد حجم الـ **component**، أولاً إختره بالفأرة. (يقوم **Delphi** تلقائياً باختيار الـ **component** الى ثم إضافتها حديثاً) وهذا يحيط الـ **component** بمربعات صغيرة، تسمى **handles**. لتغيير حجم الـ **component** المختار، اضغط واسحب الـ **handles** لتحريك الـ **object** على الـ form اضغط على الـ **component** واسحب الفأرة.

- اذا تم إختيار الـ **component** (له **visible handles**) ولكن لا تستطيع استخدامه فعلى سبيل المثال لو ضغطت **Delete** نجد أنها لا تؤدي الوظيفة المطلوبة-

اضغط على ال form لتنشيطه، ثم حاول ثانية. إننى احب ان استخدم مفتاح F12 لتنشيط ال form.

● اضغط واسحب الفأرة على ال form لتكون مستطيل حاول ان تجعل ال component يقع داخل المستطيل وأترك الفأرة سوف تجد ان كل ال component التى تقع داخل المستطيل قد أصبح لها handles. يمكنك عندئذ الضغط والسحب على أى object من objects التى لها handles لتحريكها كلها. يعرض Delphi dim handles حول كل component objects المختارة. لا يمكنك استخدام هذه ال handles لتغيير حجم ال objects- يمكنك تغيير حجم ال component object كل على حده.

● وهناك طريقة اخرى لاختيار ال objects اضغط الأول لاختياره وأبق ضاغطاً مفتاح Shift وانت تضغط مؤشر الفأرة على الآخرين اختبر Edit>Select All اذا كنت تريد اختيار كل ال component objects ووضعها على ال form.

● ومع العديد من ال objects المختارة، اضغط أى واحدة منها لتوقف اختياره. فى بعض الحالات، لكى تختار مجموعة كبيرة من ال objects، من السهل ان تختارهم جميعاً ثم توقف اختيار ما لا تريده منها.

● لايقاف إختيار ال objects، اضغط الفأرة على خلفية ال form، أو مع إختيارك ل forms أخرى أو اضغط Esc (هذه طريقة جيدة للاستخدام عندما يملأ ال component objects أو أكثر ال form، مما يصعب أو يستحيل معه ضغط خلفية النافذة). ويتم إيقاف إختيار جميع ال component objects عندما لا ترى اية handles من المهم ان تعرف هذه الحالة لانه بإمكانك ان تعدل ال events وال properties الخاصة ل form فقط عندما لا يكون هناك component objects مختارة.

● لمحاذاة component objects متعددة بصورة مرتبة، اخترها واختر أمر ال Edit/Align، الذى يعرض dialog box. استخدم الازرار الموجودة فى هذا dialog لمحاذاة الحواف ولضبط المسافات بطريقة مساوية فى المواضع المخصصة، ولإعادة ترتيبات أخرى. جرب الاختيارات الموجودة لتعتاد على اوامر المحاذاة.

● عندما يغطى احد ال component objects على الآخر. استخدم Edit/Bring to Front و Edit/Send to Back لتغيير ترتيب العرض النسبى للـ

## الباب الثاني : معلومات حول الـ Visual Components

objects . لتجربة هذه الأوامر ، اصف CheckBox objects قليلة على الـ form ، ثم أصف GroupBox ، اذا قمت بتحريك الـ GroupBox فوق الازرار ، فإنه يغطيها . ولكي تجعل الازرار تظهر مرة أخرى ، اختر الـ GroupBox واختر Edit|Send to Back لتنقله خلف الازرار . (قد تجد ان شيئاً لم يحدث ، فقد يكون السبب هو انك اخترت الازرار وليس الـ GroupBox . لاختيار الـ GroupBox فقط ، أولاً قم بإلغاء إختيار جميع الـ objects ، ثم اضغط الـ GroupBox .

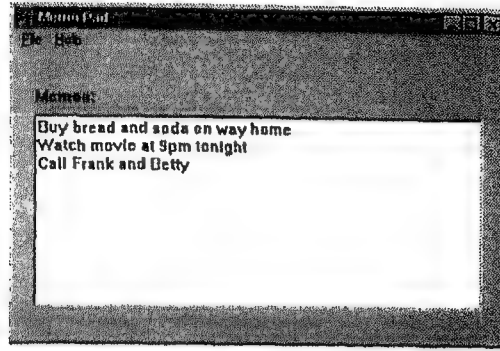
● اثناء التشغيل ، يستطيع المستخدمون ضغط مفتاح Tab لتحويل الـ Focus من احد الـ controls الى الآخر . (يرجع الـ Focus الى control الحالى الذى يتلقى أى تعامل معه من لوحة المفاتيح) . لضبط التعامل مع tab order اختر بعض أو كل الـ component objects واختر أمر الـ Edit|Tab Order . وهذا يظهر dialog box ويماكنك استخدامه لتحديد الترتيب الذى بموجبه يصبح كل control مستعداً لإستجابة الضغط على مفتاح Tab . ان أوامر الـ dialogs بديهية - ببساطة اضغط واسحب اسماء الـ component objects لإعادة ترتيبها أو اختر اسماً واضغط الزر الاسفل للـ dialog أو الزر الأعلى له .

عندما تختار اثنين أو أكثر من الـ objects للـ component ، تظهر نافذة الـ Object Inspector الخصائص والـ events المشتركة فيما بينهما . وأى تغيير فى هذه القيم تؤثر على كل الـ objects المختارة . على سبيل المثال لتغيير text style لمجموعة من check boxes أولاً ، اخترها جميعاً ثم حدد أختيارك الخاصة الـ form المشتركة .

**Tip** فكرة: هل تعرف انك تستعرض الـ events الخصائص المشتركة عندما تكون قائمة اللائحة الخاصة بالـ Object Inspector فارغة .

### المثال التطبيقي MemoPad

والمثال الخاص بهذه الفقرة ، وهو الـ MemoPad ، يوضح كيفية استخدام الـ Label و Memo . ويظهر البرنامج ايضاً كيفية إنشاء menu bar للأوامر ويوضح شكل (٢-٢) ظهور الـ MemoPad .



شكل (٢-٢)، MemoPad يعرض ال components Label , Memo

### تشغيل ال MemoPad:

لتشغيل تطبيق ال MemoPad الآن، استخدم File|Open..... . أو اضغط زر Open-project لفتح ال Memopad.dpr. اضغط F9 لتشغيل البرنامج. اكتب ملحوظات قليلة، ثم اترك ال MemoPad وارجع الى Delphi حتى تستطيع فحص ال components وال form الخاصة بالتطبيق.

اختر امر ال File|Save الخاص بال MemoPad لحفظ مذكرات في text file يحمل اسم Memos.txt في الدليل الحالي. ودائماً ما يستخدم ال MemoPad اسم الملف هذا اختر File|Exit لإنهاء ال MemoPad، والذي أيضاً يؤدي الى حفظ مذكراتك تلقائياً. (يمكنك أيضاً استخدام اية طريقة أخرى لترك البرنامج). ان اختيار Help|About يعرض ال display للرسالة الموضحة في شكل (٢-٣)، والذي يعرف التطبيق.



شكل (٢-٣)، dialog خاص بالرسالة الخاصة

بال MemoPad تعرض عندما تختار امر Help|About بالبرنامج

ان اثنين من فوائد الأوامر الخاصة بالمثال ولتلك الأوامر accelerator keys يمكن للمستخدمين ان يضغطوها لإختيار أوامر القائمة دون استخدام الفأرة. اضغط

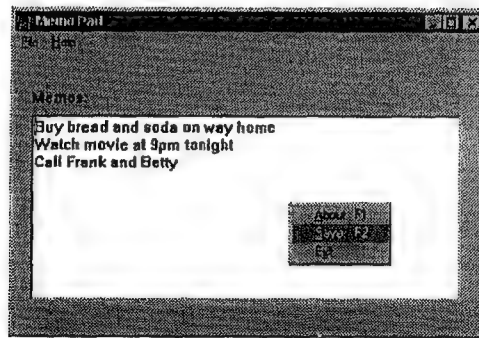
## الباب الثاني : معلومات حول الـ Visual Components

F2 لحفظ مذكراتك . اضغط F1 لعرض display الخاص للرسالة . بالإضافة الى ذلك ، فإن اوامر القائمة تضع خطة تحت المفاتيح الهامة . على سبيل المثال ، اضغط Alt+F لاختيار FileExit.

**ملحوظة:** ان Delphi يقوم بإعطاء مصطلح خاطئ بعض الشيء لكل مفاتيح الـ shortcut في لوحة المفاتيح . من الناحية التقنية ، يعتبر الـ accelerator هو مزيج من واحداً أو أكثر من الـ Ctrl ، أو الـ Alt أو مفاتيح الوظائف الأخرى المخصصة لأمر قائمة والموضحة الى اليمين منها في menu's pop-up window و shortcut key هو حرف موضوع تحته خطأ في قائمة أو امر القائمة .

وهذه ليست مشكلة كبيرة - فقط كن حذراً من ان المفاتيح التي تحتها خط الخاصة بأمر القائمة ، مثل الـ Alt+F X ، تختلف عن الـ ccelerator keys مثل F1 أو Ctrl+X . وأي امر قائمة يمكن ان يكون له أى من النوعين السابقين أو كلاهما .

ويستخدم برنامج الـ MemoPad ايضاً PopupMenu component لتوفير بديل آخر لتنفيذ الأوامر حرك مؤشر الفأرة في أى مكان داخل نافذة الـ MemoPad - ولكن ليس على menu bar واضغط زر الفأرة الايمن . كما يوضح شكل (٢-٤) فإن هذا يفتح pop-up menu صغيرة .



شكل (٢-٤): pop-up menu الخاص بـ MemoPad  
تظهر عندما تضغط زر الفأرة الأيمن

تعمل كالقائمة الرئيسية ولكن تظهر عند موضع الفأرة . لإغلاق نافذة القائمة هذه ، اضغط زر الفأرة الايمن خارج pop-up menu أو اختر أمراً (أو اضغط Esc) .



**Tip** فكرة: إعطى المستخدمين دائماً methods مختلفة كثيرة لتنفيذ الأوامر العامة. بهذه الطريقة، يمكن للمبتدئين اختيار أوامر من قوائم التطبيق (وهي بسيطة نسبياً للتعود على كيفية استخدامها)، ولكن الخبراء يمكنهم ان يحفظوا shortcut keys ويستخدموا pop-up menu لتنفيذ الأوامر بسرعة.

### إنشاء الـ MemoPad

بعد ان تعتاد على تطبيق الـ MemoPad، استخدم Delphi لفحص الـ components و الـ forms للبرنامج. اخرج من الـ MemoPad الآن لتعود الى Delphi. اختر كلاً من component objects للبرنامج وبالفأرة أضغط على الـ objects (الزر الأيسر)، واستعرض الـ events وخصائصها في الـ Object Inspector.

**تحذير:** تعود على ان تضغط مرة واحدة على الـ component objects التي على الـ form ففي كثير من الـ components اذا ضغطت مرتين عليها بالفأرة، فإن Delphi يقوم بإنشاء procedure فارغ في الـ source code للبرنامج للـ events الافتراضي، والذي هو عادةً OnClick. اذا حدث هذا مصادفة، يمكنك اختيار Edit/Undo لاستعادة أو مجرد تجاهل الادخال. واذا قمت مصادفة بإنشاء event handler، لا تريده، فلا تقلق- مادمت لم تضيف اية عبارات أو تعليقات إلى الـ procedure، فإن Delphi يحوها تلقائياً في المرة التالية التي تقوم فيها بتنفيذ المشروع.

استعرض ايضاً العبارات الموجودة في Main unit (النافذة التي تحمل العنوان Main.pas). اذا لم تعثر على هذا النص، قم باظهار نافذة code editor الـ View/Units أو اضغط زر Toggle Form/Unit. أو، يمكنك ضغط F12 مرة واحدة أو عدة مرات حتى تظهر النافذة. يحتوي Main.pas على source code للـ Object Pascal، والتي يطابقها Delphi مع الـ form والـ visual component objects الخاصة بها. واثناء تطوير التطبيق، فسوف تنتقل غالباً بين الـ form و الـ source code التابع لها.

من المفيد ان تفكر في الـ form و الـ unit التابع لها على انهما يقدمان شكلان مختلفان لنفس الـ objects، وتوضح الـ form المظهر الخارجى للـ controls

## الباب الثاني : معلومات حول الـ Visual Components

editor code تعرض الـ Object Pascal Commande والتي تؤدي انشطة الـ components مثل الاستجابة الى ضغطة زر ما أو حفظ بيانات في ملف. وكل form لديها unit واحدة (والتي قد تستخدم units أخرى) قد تحتوي على procedures و functions وتعريفات أخرى متعددة للـ component objects التابعة للـ form.

لا تهتم بفهم كل البرمجة الموجودة في ملف الـ Main.pas الخاص بالـ MemoPad، والموضح هنا في القائمة (١-٢). ان هذا الملف وغيره من الملفات الموجودة في هذا الكتاب قد تم وضعها في قوائم للرجوع إليها كمراجع فقط. ان Delphi قد أنشأ معظم البرمجة الموضحة هنا - لبرمجة مهام التطبيق، ولقد قمت بكتابة عبارات داخل الـ procedure declarations التي قام Delphi بإدخالها في الـ component editor للتطبيق. وتعتبر هذه واحدة من خصائص Delphi الرئيسية للتطوير السريع للتطبيق - لإنشاء تطبيق، اختر component ثم إملاء حرفياً الفراغات التي يقدمها Delphi.

### القائمة (١-٢): الـ Memopad/Main.pas

unit Main;

interface

uses

Windows, SysUtils, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, Menus, StdCtrls;

type

TMainForm = class(TForm)

MainMenu1: TMainMenu;

PopupMenu1: TPopupMenu;

File1: TMenuItem;

```

Save1: TMenuItem;
Exit1: TMenuItem;
Help1: TMenuItem;
About1: TMenuItem;
About2: TMenuItem;
Save2: TMenuItem;
Exit2: TMenuItem;
MemoLabel: TLabel;
Memo1: TMemo;
procedure Save1Click(Sender: TObject);
procedure Exit1Click(Sender: TObject);
procedure About1Click(Sender: TObject);
procedure FormActivate(Sender: TObject);
procedure FormClose(Sender: TObject);
var Action: TCloseAction);
    private
{ Private declarations }
    public
{ Public declarations }
end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

```

## الباب الثاني : معلومات حول الـ Visual Components

---

```
procedure TMainForm.Save1Click(Sender: TObject);
begin
    Memo1.Lines.SaveToFile('memos.txt');
end;
```

```
procedure TMainForm.Exit1Click(Sender: TObject);
begin
    Close;
end;
```

```
procedure TMainForm.About1Click(Sender: TObject);
begin
    MessageDlg(
'Memo Pad'#13#10' ÷ 1995, 1998 by TomSwan'#13#10'Version 1.00',
    mtInformation, [mbOk], 0);
end;
```

```
procedure TMainForm.FormActivate(Sender: TObject);
begin
    if FileExists('memos.txt')
        then Memo1.Lines.LoadFromFile('memos.txt')
        else Memo1.Lines.SaveToFile('memos.txt');
end;
```

```
procedure TMainForm.FormClose(Sender: TObject;
    var Action: TCloseAction);
begin
    Memo1.Lines.SaveToFile('memos.txt');
end;
```

الآن، حاول ان تعيد إنشاء تطبيق ال Memopad بنفسك . اذا تعثرت، استخدم الملفات الموجودة على القرص المدمج فى دليل ال Source\MemoPad كمرشد لك، أو إرجع للقائمة (١-٢). إتبع هذه التعليمات خطوة بخطوة:

١- استخدم ال Windows Explorer لإنشاء دليل مثل C:\Projects\Memopad لحفظ ملفات تطبيقك. اذا لم تكن تريد حفظ التطبيق، يمكنك تخزين الملفات فى دليل يحمل اسم Temp واحذفها فيما بعد.

٢- إبدأ مشروع جديد. إضغط داخل ال form لتختارها، وقم بتغيير خاصية ال Caption فى نافذة ال Object Inspector لتصبح ال Memopad. من الآن فصاعداً، عندما اذكر تغيير خاصية، اختر أولاً ال form أو component، واذا لزم الأمر، اضغط ال Properties tab فى نافذة ال Object Inspector. ثم اختر الخاصية وغير قيمتها، الموضحة الى اليمين من اسم الخاصية.

٣- قم بتغيير خاصية Name لل form لتصبح MainForm. وهى كلمة واحدة بلا مسافة بينها اذا كتبت رموز لا يسمح بها داخل خاصية Name، فإن Delphi يعرض رسالة خطأ. يجب ان تبدأ الاسماء بحرف أو هذا الرمز (-)، ويمكن ان تحتوى على حروف أو ارقام أو هذا الرمز (-) فقط. ان ال Name الذى تقوم بإدخاله يتماثل مع الاسم المبرمج لل object's ولذا، فإن فى ال source code يمكنك الإشارة لل Form object عن طريق البرمجة من خلال ال main form.

٤- اختر File\Save All، أو اضغط زر Save All لإنشاء ملفات القرص الخاصة بالمشروع. يمكنك الانتظار الى وقت لاحق لحفظ المشروع، ولكن ان تفعل هذا الآن يؤدى الى إنشاء اسماء ملفات يستخدمها Delphi لإنشاء عناصر متنوعة فى نص البرنامج. إننى انصحك ان تحفظ البرنامج باسرع ما يمكن بعد ان تطلق اسماً على ال form الرئيسية به.

٥- ان Delphi يقدم لك أثنان من ال file dialogs، واحداً تلو الآخر. فى ال dialog الأول، والذى يحمل العنوان Save Unit1 As، تحول الى الدليل من الخطوة رقم (١)، واكتب Main فى مربع ال File Name. اضغط Enter أو اضغط OK. فى ال dialog الثانى، الذى يحمل العنوان Save Project1 As،

## الباب الثاني : معلومات حول الـ Visual Components

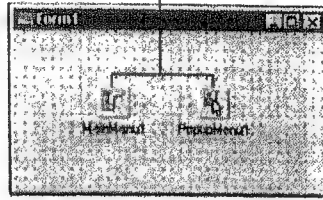
اكتب الـ MemoPad بكأسم الملف . اضغط Enter أو اضغط OK . لقد اتممت إنشاء ملفين الآن : Main.pas و Memopad.dpr . يقوم Delphi بإلحاق امتدادات الـ .pas والـ .dpr لاسماء ملفاتك . ولإجراء الاختيار بشكل سريع ، يمكنك استخدام الاسماء الافتراضية حسب النظام وهي Project1 و Unit1 ، لكن فى أغلب الحالات ، سوف تريد تغييرها لتصبح اسماء ملف وصفية .

**فكرة:** يجب ان تكون اسماء ملف المشروع والـ Units تتبع القواعد الصحيحة فى الـ Pascal . ان اسماء الملفات الطويلة تكون مقبولة ، ولكنها قد لا يكون لديها مسافات بين الحروف أو رموز أخرى مثل علامات الترقيم .

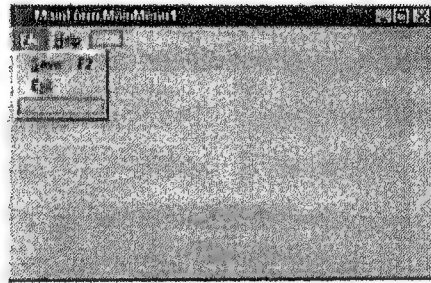
٦- أضف MainMenu component ، على الـ form . أضف أيضاً PopupMenu component . كلاهما متواجدان فى Standard palette . فى هذا المثال ، استخدم أسماء الـ object المتاحة ، MainMenu1 و PopupMenu1 ، للتعرف على الـ objects . ولأنها components معقدة نسبياً ، فإن Delphi يعرضهما فى الـ form كايقونات تمثل مظهرهم النهائى [انظر الشكل (٢-٥)] . وتكون الايقونات غير مرئية اثناء وقت التشغيل . قم بتحريك كلتا الايقونتين الى الركن الايمن العلوى من الـ form ، أو الى أى مكان آخر ملائم . اذا لم ترى labels للأثنين كما هو موضح فى الشكل ، اختر ToolsEnvironment Options ، اضغط Preferences page tab ، وقم بإختيار الـ check box لعرض الـ component captions فى الـ Form Section designer . ويظهر الـ Captions فقط للـ components مثل الـ MainMenu و PopupMenu التى ليس لها labels أو شكل معين تظهر به . فالازرار ، على سبيل المثال ، والتى لها labels وهى من النوع الذى ترى أن له شكل مميز عن بقية الـ components وليس له أى Captions .

٧- لإنشاء اوامر القائمة ، اضغط مرتين MainMenu1 component object ، والذي يفتح الـ Menu Designer الخاص بـ Delphi ، كما هو موضح فى شكل (٢-٦) . لإنشاء قائمة الـ File ، اكتب &File ، واضغط Enter لتغيير خاصية الـ Caption لشكل القائمة هذا . اكتب حرفاً مسبقاً بعلامة (&) لتحديد Alt shortcut key الخاص بالامر .

These objects are not visible at runtime



شكل (٥-٢)؛ ايقوناتي ال MainMenu و PopupMenu component يظهران فقط في وقت التصميم - ولا يظهران في نافذة البرنامج أثناء تشغيله. وتعليقات الايقونة الموضحة هنا تعتبر اختيارية



شكل (٦-٢)؛ هذا هو ال Menu Designer التابع لـ Delphi موضحاً  
الاورامر الموجودة في ال MainMenu object الخاص بال MemoPad

٨- يقوم ال Menu Designer الآن بتحديد المكان التالي لإدخال أمر ، في هذه الحالة ، تحت قائمة ال File . أدخل Save & لإنشاء امر ال Save . لا تضغط Enter (إذا فعلت هذا ، استخدم مفتاح السهم المشير الى أعلى لإعادة تحديد الأمر) . اختر خاصية ال ShortCut في نافذة ال Object Inspector ، اضغط السهم المشير الى اسفل ، وأختر key shortcut F2 accelerator لهذا الأمر .

Tip فكرة: من الأسرع ان تكتب F و 2 وتضغط Enter بدلاً من ان تبحث عن مفاتيح تستخدم قائمة اللائحة الطويلة هذه التابعة لهذه الخاصية .

٩- اضغط المساحة الفارغة تحت أمر ال Save في ال Menu Designer ، واختر خاصية ال Caption في نافذة ال Object Inspector . ادخل E&xit لإنشاء مفتاح ال Exit وتعيين الحرف X Alt shortcut key الخاص بالأمر .

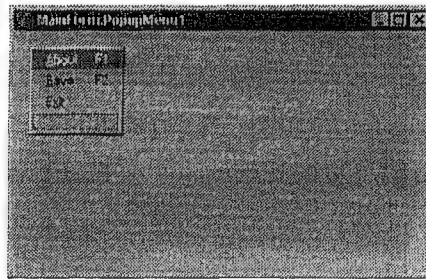
## الباب الثاني : معلومات حول الـ Visual Components

**فكرة:** عند إدخال أوامر قائمة جديدة، اختر دائماً خاصية الـ Caption قبل ان تكتب نص الأمر.

١٠- باستخدام اساليب مماثلة، ادخل قائمة الـ Help وأمر الـ About الخاص بها. ويكون F1 من accelerator shortcut key للأمر بنفس الطريقة التي عينت بها F2 للأمر Save.

١١- اغلق نافذة الـ Menu Designer لترى قائمتك في الـ form. يمكنك فحص هذه القائمة، ولكن في الوقت الحالي، لا تختار أمراً، مما يؤدي لإنشاء event handler في الـ unit module. (ولا يوجد ضرر اذا فعلت هذا من دون قصد).

١٢- اضغط مرتين الـ PopupMenu component لتعيد فتح الـ Menu Designer، والذي يعرض هذه المرة نمط، أبسط للـ floating pop-up menus [انظر شكل (٧-٢)]. أضف أوامر الـ Exit والـ Save الـ About في هذه القائمة. ضع علامة & قبل الحروف التي تحتها خط. استخدم خاصية الـ ShortCut لكل أمر لتحديد accelerator shortcut keys مثل F1 و F2.



شكل (٧-٢): الـ Menu Designer الخاص بـ Delphi موضحاً الأوامر الموجودة في الـ PopupMenu object الخاصة بالـ MemoPad

١٣- لا يكفي مجرد إنشاء PopupMenu component يجب عليك أيضاً أن تخبر الـ form كيفية استخدامها اختر نافذة الـ Menu Designer، ثم اضغط الـ form لإيقاف اختيار أي component يجب أن ترى الـ MainForm: TMainForm في قائمة اللائحة الخاصة بالـ Object Inspector. قم بتغيير خاصية الـ PopupMenu التابعة للـ form لتصبح PopupMenu 1. ويمكنك كتابة هذا الاسم أو اختياره من



قائمة اللائحة لهذه الخاصية. وهذا يجعل ال from تعرض 1 PopupMenu component عندما يضغط المستخدم زر الفأرة الأيمن على ال from .

١٤ - اضغط F9 لتشغيل التطبيق ، والذي لم يتم بعد حتى هذه المرحلة . إننى فى الغالب أفعل هذا مباشرة بعد إنشاء أوامر برنامج حتى أتمكن من اختبار مظهرهم . حاول تجربة أوامر قائمة البرنامج (لا أحد منهم يعمل حتى هذه المرحلة) ، واضغط زر الفأرة الأيمن لعرض floating pop-up menu والتي لا تعمل هى الأخرى . للعودة الى Delphi ، اضغط Alt+F4 ، اضغط مرتين زر قائمة النظام ، أو اضغط زر إغلاق ال Windows 95 .

لقد أتممت الآن المرحلة الأولى من برمجة ال MemoPad . بعد أن تسترح قليلاً ، قم بتحميل مشروع ال MemoPad ، إذا لزم الأمر ، واتبع الخطوات التالية لإتمام التطبيق :  
١ - أضف ال Label component على ال form ، وقم بتغيير الخاصية Name لتصبح MemoLabel . اضغط Enter ولاحظ أن Delphi يغير ال Caption تلقائياً تبعاً لتغير اسم ال component .

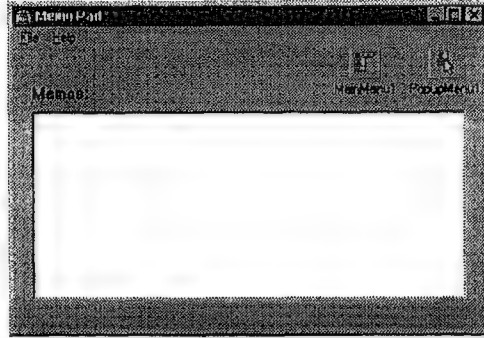
٢ - فى الغالب يكون من غير المستحب لل component أن يظهر اسمه الداخلى . لتغيير النص المعروف الخاص بال labels ، اختر ال MemoLabel وغير خاصية ال Caption لتصبح Memos :

٣ - قم أيضاً بإضافة ال Memo component على ال form . فى هذه المرة ، يمكن أن تترك خاصية ال Name تحدد حسب نظام البديل الافتراضى للبرنامج ، وهو Memo1 . رغم ذلك ، قم بحذف كل الرموز من خاصية ال Lines حتى تصبح مساحة الإدخال خالية . لكى تفعل هذا ، اختر خاصية ال Lines لل Memo1 ، واضغط الزر البيضوى على اليمين . هذا يفتح محرر قائمة String الخاصة بـ Delphi ، والتي يمكنك استخدامها لحذف السطر Memo1 (اضغط المساحة الخلفية عدة مرات متتالية أو قم تحديد هذا النص واضغط Del) اضغط Enter أو اختر زر OK ال editor لتجعل التغيير ثابتاً .

٤ - قم بإعادة تحديد حجم ال Memo1 ورتب كل ال components لتجعل العرض الخاص بك يتلاءم مع شكل (٢-٨) . فى هذه النقطة ، قد تريد أن تحفظ المشروع وتشغله لترى المظهر النهائى للتطبيق .

## الباب الثاني : معلومات حول الـ Visual Components

**فكرة:** تستطيع أن تجرب تطبيقاتك . يمكنك أن تضغط F9 لتشغيل البرنامج في أى مرحلة من مراحل تطويره .



شكل (٢-٨): استخدم هذه الصورة للـ MemoPad form  
فى Delphi كمرشد لك عندما تقوم بإنشاء تطبيق

٥- لبرمجة أوامر الـ MemoPad، اختر أمر الـ FileSave بالبرنامج فى الـ form . (لا تختار أمراً فى تطبيق أثناء تشغيله - اختر من خلال الـ form) عندما تفعل ذلك ، يقوم Delphi بإضافة procedure بعيد عن الـ event handler من خلال الـ source code، وينتقل المؤشر تلقائياً الى الـ editor . وهناك ، يمكنك أن تكتب عبارة لأداء أعمال الأمر . ادخل العبارة التالية بين البداية والنهاية . ويجب أن يكون النص الواقع بين قوسين تحده علامات تنصيص فردية . لا تستخدم العلامة الزوجية . حول هذه البرمجة وغيرها من البرمجة المقترحة فى هذا الكتاب، إرجع الى ملفات أو قوائم القرص الخاصة بالمشروع [أنظر القائمة (٢-١) لهذا المثال] .

```
Memo1.Lines.SaveToFile('memos.txt');
```

٦- وبالإنجليزية ، فإن العبارة التى أدخلتها لتوك تحفظ سطور الـ Memo1 فى ملف يسمى Memos.txt . والـ Lines هو عبارة عن object يمتلكه Memo1 component object بمعنى ، إن Lines هو نفسه object داخل الـ component والـ SaveToFile وهو عبارة عن method فى هذا الـ object يمكن لـ Lines أن يؤديه . والنقاط ، وتسمى dot notation ، تحدد أن Memo1 ، Lines ، و SaveToFile مرتبطة ببعضها . ويمثل المقطع 'memos.txt' اسم ملف . والعبارة تقرر المقطع الموضح بين الأقواس الى الـ SaveToFile method . لاحظ أن العبارة

تنتهى بفصلة منقوطة . من الناحية الفنية ، وفى Pascal ، تفصل الفصلة المنقوطة إحدى العبارات عن الأخرى . فى هذه الحالة ، لأنه لا يوجد عبارة أخرى تالية ، يمكنك ترك الفصلة المنقوطة ؛ رغم ذلك ، فلا ضرر من أن تضعها . اذا وضعت الفصلة المنقوطة فى موضع خاطئ ، أو اذا لم تضع واحدة هامة ، فإن Delphi يخبرك بهذا فى رسالة خطأ عندما تقوم بتشغيل البرنامج .

٧- اكمل اوامر ال MemoPad الأخرى . اختر ال FileExit فى ال form . أدخل Close; بين كلمة البداية الرئيسية وكلمة النهاية الرئيسية (إرجع الى TMainForm.ExitClick in Main.pas procedure) . كما عرفت فى الباب السابق ، ان ال Close procedure ، واحد من ال Delphi procedures التى لا تخصى والتى يمكنك إستدعاءها ، لتغلق النافذة . فى هذا المثال ، إذا كانت هذه النافذة الرئيسية للبرنامج ، فإن Close أيضاً ينهى البرنامج .

٨- إرجع الى ال form ، واختر HelpAbout . ادخل العبارة التالية بين البداية والنهاية . اكتب بحذر كل حرف بالضبط كما هو موضح ، أو اذا أردت ان تغش ، انسخ والصق العبارة من ملف ال Main.pas . (اننى أؤيد الغش دائماً ، فلا تتردد فى ان تنسخ من هذه الصفحات أو من القرص المدمج المرفق . إنك لا تحتاج الى إذن خاص لتستخدم ال source code (بهذا الكتاب فى عملك) . اكتب كل ما يلى فى سطر واحد - إنه مقسم على سطرين هنا بسبب حدود المسافة :

```
MessageDlg('Memo Pad'#13#10'@cw 1995,1998 by Tom
Swan'#13#10'Version 1.00', mtInformation, [mbOk], 0);
```

٩- ان العبارة تنشئ message dialog يعرض ملحوظة حق الطبع لل MemoPad ورقم النسخة . وفى MessageDlg procedure ، يمرر البرنامج اربعة معاملات بين قوسين ، تفصلها فصلة . والمعامل الأول هو مقطع ، تجده علامتى تنصيب . فى هذا المقطع ، تقوم مجموعة العلامات #13#10 بادخال ال carriage return و ASCII control codes لجعل النص الذى يكتب بعدها تكتب فى سطر جديد . لإدخال رمز حق الطبع ، استخدم Windows Character Map utility ، أو قص والصق الرمز من الملفات الموجودة على القرص المدمج . (أو ، مجرد ان تكتب C) . والمعامل ال mtInformation للاختيار من بين عدة

## الباب الثاني : معلومات حول ال Visual Components

أنماط dialog styles ، والمعامل [mbOk] يعتبر مجموعة والتي لها في هذا المثال عضو واحد ، هو mbOk ، يحدد نمط ال dialog بواسطة زر OK . والمعامل 0 يعتبر شاغل المكان dialog's help context والذي لا يستخدم ال MemoPad . مرة أخرى ، تنهى الفصلة المنقوطة العبارة . (لمزيد من المعلومات حول قيم معاملات MessageDlg ، لمزيد من المعلومات عن ال procedure إرجع الى Delphi's (online help).

١٠- لبرمجة اوامر pop-up menu ، PopupMenu1 object (الذي عليه سهم) الذي قمت بإضافته على ال form وهذا يعيد فتح نافذة ال Menu Designer . اختر Events page tab في نافذة ال Object Inspector . اختر امر ال About في ال Menu Designer ، وحددOnClick event بـ About1Click . يمكنك إدخال اسم ال event handler هذا ، اذا تم اختياره من قائمة لائحة ال OnClick . حدد ال OnClick event الخاص بأمر ال Save بـ Save1Click . حدد ال OnClick event الخاص بأمر ال Exit بـ Exit1Click . إنها نفس برامج event handler التي قد برمجتها للقائمة الرئيسية- وهي إظهار لكيفية مشاركة أكثر من events في نفس ال code .

١١- اغلق نافذة ال Menu Designer . اضغط داخل خلفية ال form ، واذا كان ضرورياً ، اختر Events page tab في نافذة ال Object Inspector . اضغط مرتين بالفأرة المساحة الخالية الى اليمين من ال OnActivate event ، وادخل بعد ذلك عبارة ال if-then-else الواقعة بين البداية والنهاية . وتعطى العبارة نص الملف . المعين في component's Lines object :

```
if FileExists('memos.txt')
then Memo1.Lines.LoadFromFile('memos.txt')
else Memo1.Lines.SaveToFile('memos.txt');
```

والعبارة السابقة تستدعي ال FileExists Function ، والتي ترجع True اذا لم يكن . اذا كان الملف موجوداً ، فإن العبارة تعطى الملف في ال Lines الخاصة بال Memo1 . اذا لم يكن الملف موجوداً ، فإن العبارة تحفظ Lines في ملف جديد . ولأنه خاصية ال Lines التابعة لل object تعتبر فارغة في هذه النقطة ، فإن

استدعاء SaveToFile method الخاص به ينشئ ملف جديد خالى . بالرغم من أنه قد يبدو أن كثير من العمليات تحدث ، إلا أن بناء الـ if-then-else بأسره يعد عبارة واحدة ، فإن فصلة منقوطة واحدة تلزم لفصل هذه العبارة عن غيرها . اذا وضعت فصلة منقوطة فى نهاية السطر الثانى فإن Delphi يعرض رسالة خطأ عندما تقوم بتشغيل البرنامج .

١٢ - اختر الـ form (من المحتمل أن يكون قد تم اختيارها) ، واضغط مرتين المسافة الواقعة الى اليمين من OnClose event . يقوم Delphi بإضافة procedure لهذا الـ event والذي يحدث عندما تغلق نافذة البرنامج . أدخل العبارة التالية بين البداية والنهاية لحفظ components Lines فى ملف مذكور :

`Memo1.Lines.SaveToFile('memos.txt');`

١٣ - اضغط F9 لتشغيل البرنامج حاول تجربة الأوامر وارجع الى Delphi . قارن ملف الـ Main.pas للبرنامج الخاص بك بالملف الذى يحمل نفس الاسم على القرص المدمج الخاص بهذا الكتاب . قد تكون بعض الـ procedures فى ترتيب مختلف اعتماداً على الترتيب الذى أدخلت به الـ components فى form والترتيب الذى برمجته الـ event handlers ، ولكن يجب أن تكون العبارات والشكل العام هو نفسه .

## :Additional Components و Dialogs Components

فى هذه الفقرة ، تقوم بتطوير تطبيق لعرض الـ bitmaps الذى يستخدم الـ visual components من فئتين مختلفتين لـ Delphi ، وهما Additional ، و Dialogs ، اختر الـ page tabs المخصصة لهما واستخدم hint boxes لإيجاد مكان الـ mentioned المذكورة هنا . (تذكر : ضع مؤشر الفأرة على أيقونة component وانتظر ظهور hint boxes) .

من بين العناصر الأخرى ، تقدم فئة الـ Additional component Graphics التى يمكنك استخدامها لإنشاء bitmap و Graphics Device Interface (GDI) مثل المستطيلات والدوائر والخطوط . والـ Windows GDI method تعين الـ device-independent لرسم وطباعة الرسومات ويمكنك التوصل لـ GDI بإضافة component objects على الـ form .

## الباب الثاني : معلومات حول الـ Visual Components

وتقدم الـ Dialogs فئة الـ components التى تكون الواجهة بين الـ Comman Windows و dialog Boxes. بالرغم من ان هذه هى كل Delphi component ، يقدم الـ Windows الـ code والمظهر الفعلى للـ dialog الناتجة. ان استخدام الـ dialog العامة يعطى للمستخدمين طرق مألوفة لتبادل دلائل الاقراص ، واختيار اسماء الملف ، اختيار الـ fonts والالوان ، والبحث عن أو استبدال القيم فى الوثائق.

### استخدام الـ Image component :

لإنشاء الـ bitmap على الـ form أضف الـ Image component object من فئة الـ Additional جرب هذا الآن. اختر أولاً File\New Application لإخلاء مساحات العمل فى Delphi ، ثم أضف الـ Image component object على form فارغة ولان الـ Image object ليس له run-time shape ولأن محتوى الـ bitmap يعرف مظهره- يعرض Delphi الـ Image component object شفافاً الذى لا يظهر فى البرنامج النهائى وباختيار الـ object outline Image ، قم باختيار خاصية الـ Picture فى نافذة الـ Object Inspector ، واضغط الزر البيضوى لفتح الـ Picture Editor الخاص بـ Delphi. أو ، يمكنك ان تضغط مرتين على الـ component object. استخدم الـ Picture Editor لإدخال أى bitmap داخل التطبيق. أولاً ، اضغط زر Load ثم اختر ملفاً ينتهى بالامتداد الـ bmp ، (bitmap) ، ico ، (icon) ، (Windows .wmf) (metafile) ، أو emf (enhanced metafile) . يمكنك حينئذ ضغط الـ Save لنقل الملف على ملف قرص آخر ، أو الـ Clear لمسح الصورة التى تم تحميلها سابقاً ، أو الـ Cancel لإلغاء أداء عمليات ، أو الـ OK لتحميل الصورة فى Image component object .

إذا كنت مستمراً ، استخدم زر الـ Load للتعود الى دليل الـ Data ، على القرص المدمج . إفتح الايقونة Sample.ico ، واضغط OK . تظهر ايقونة الملف فى Image component الخاص الـ form ، وتتغير خاصية الـ Picture لتصبح TIcon . أو ، لإدخال (bitmap) ، اختر ملف Sample.bmp يمكنك أيضاً تحميل صور أخرى اذا كان لديك بعض (bitmaps) مخزنة على القرص . قد يجب عليك

إعادة تحديد حجم Image object لترى (bitmap). ولتفعل هذا تلقائياً، حدد خاصية الـ Image AutoSize لـ True. يمكنك إعادة تحديد حجم bitmap's قم بتحميلها داخل Image component object، ولكن تبقى صورة الايقونة دائماً ثابتة الحجم.

**فكرة:** بعد تحميل الـ bitmap داخل الـ Image object، لتقليل أو توسيع الـ object ليلائم الـ bitmap بدقة حدد الـ AutoSize بـ True، اضغط مرتين Image object، واضغط OK فى الـ Image Editor.

Tip

## التطبيق BitView

ان التطبيق الموصوف فى هذه الفقرة يجمع بين Standard والـ Additional Dialogs لإنشاء برنامج لعرض bitmap.

### تشغيل BitView

لتجربة التطبيق، افتح ملف مشروع Bitview.dpr فى الدليل الفرعى Bitview واضغط F9. اختر أى ملف ينتهى بـ .bmp. هناك مثال اذا احتجته فى دليل الـ Data على القرص المدمج. شكل (٢-٩) يوضح الـ Bitview وهو يعمل.



شكل (٢-٩): تطبيق الـ Bitview هذا بإمكانه ان يعرض أى bitmap file سوف تقوم الآن بإنشاء تطبيق الـ Bitview الخاص بك. اذا قمت بتشغيل الـ Bitview، قم بانهاء البرنامج الآن قبل الاستمرار.

## الباب الثاني : معلومات حول الـ Visual Components

### إنشاء الـ Bitview:

توضح القائمة (٢-٢) Main.pas source code لتطبيق الـ Bitview .  
استخدم هذا الملف كمرشد لك عندما تنشئ التطبيق باتباع التعليمات الموجودة بعد  
القائمة خطوة بخطوة .

#### القائمة (٢-٢) : Bitview\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
Windows, SysUtils, Messages, Classes, Graphics,  
Controls,  
Forms, Dialogs, Menus, ExtCtrls;
```

```
type
```

```
TMainForm = class(TForm)
```

```
  BitImage: TImage;
```

```
  MainMenu1: TMainMenu;
```

```
  File1: TMenuItem;
```

```
  Open1: TMenuItem;
```

```
  Exit1: TMenuItem;
```

```
  OpenDialog1: TOpenDialog;
```

```
  procedure Open1Click(Sender: TObject);
```

```
  procedure Exit1Click(Sender: TObject);
```

```
private
```

```
{ Private declarations }
```

```
public
```



```

{ Public declarations }
end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.Open1Click(Sender: TObject);
begin
  if OpenFileDialog1.Execute then
    begin
      BitImage.Picture.LoadFromFile(OpenDialog1.FileName);
      Caption := OpenFileDialog1.FileName;
    end;
end;

procedure TMainForm.Exit1Click(Sender: TObject);
begin
  Close;
end;

end.

```

لإنشاء تطبيق الـ Bitview الخاص بك، اختر أولاً الـ File|New Project  
 لمسح أى مشروع تم تحميله حالياً سوف تعتمد على خطوات البداية بمرور الوقت، لذا  
 لن اكررها فى المستقبل. كما أننى أختصر بعض الخطوات التالية، مثل كيفية حفظ

## الباب الثاني : معلومات حول الـ Visual Components

مشروع. اذا كنت اكملت التطبيق السابق بنجاح، فلن تقابلك مشكلة مع هذا التطبيق:

١- قم بتغيير خاصية الـ Caption للـ form لتصبح Bitmap Viewer. قم بتغيير خاصية الـ Name لتكون MainForm.

٢- قم بتغيير خاصية الـ WindowState للـ form من wsNormal الى wsMaximized باختيار تلك القيمة من قائمة اللائحة بالخاصية. هذا يؤدي لعرض نافذة الـ BitView في نمط الشاشة الكاملة. وتختار القيم الأخرى حالات نافذة أخرى - يعرض الـ wsMaximized النافذة كإيقونة، ويعرض الـ wsNormal النافذة في حجمها المحدد في وضع يقرره Windows.

٣- اختر فئة الـ Additional، وأضف Image component على الـ form، غير خاصية الـ Name للـ object لتصبح BitImage. بالنسبة لموقع الـ object وحجمه لا يهم بدقة.

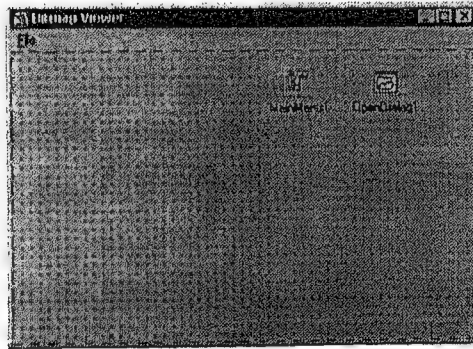
٤- ومع استمرار اختيار BitImage object، قم بتغيير خاصيته الـ Align لتصبح alClient. وهذا يؤدي الى إعادة تحديد حجم الـ component تلقائياً بنفس حجم الـ client area الخاصة بالـ form، وهي المسافة الواقعة داخل حدود النافذة.

٥- ولتجعل الـ bitmap الموجودة داخل الـ component object تتلائم أيضاً مع الـ client area وهذا يتطلب تغيير آخر. قم بتحديد خاصية الـ BitImage component's Stretch، اضغط مرتين قيمة الخاصية لتحويلها من الـ False الى الـ True. (يمكنك أيضاً كتابة أو اختيار True أو False من قائمة الخاصية، ولكن الضغط مرتين على الخاصية True/False هو أسهل طريقة لتحديد قيمتها). ومع تحديد الـ Stretch بـ True، يعرض البرنامج الـ bitmaps في حدود مساحة component. ولأن الـ component object قد تم تحديد حجمه تلقائياً وفقاً للنافذة، فكذلك تكون الـ bitmap وهذا يظهر أيضاً كيف يكون الـ component object والبيانات الخاصة به، رغم علاقة كل منهما ببعضها، في بعض الأحيان يتطلب البرمجة بشكل منفصل لكل منهما منفصلة.

٦- أضف الـ Standard MainMenu component في الـ form والاسم الافتراضي حسب نظام البرنامج، وهو MainMenu1، واصفاً بما فيه الكفاية، ولا

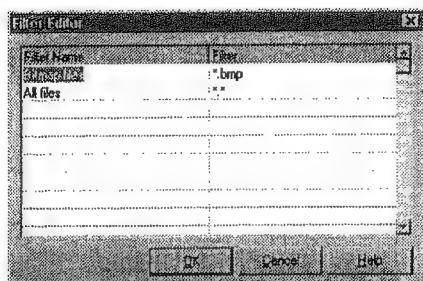
يجب عليك تغييره اضغط مرتين ايقونة menu object ، وقم بإنشاء قائمة تسمى File ذات أمرين ، Open و Exit . تذكر ان تكتب علامة ال & قبل ال shortcut key ، على سبيل المثال &File . اذا كان لديك مشكلة فى إنشاء قائمة البرنامج ، راجع خطوات ال MemoPad . لا يجب عليك تخصيص accelerator-shortcut keys ، ولكن تستطيع ان تفعل هذا اذا اردت . اغلق نافذة ال Menu Designer . عندما ينتهى .

٧- من ال Dialogs component palette ، اضع OpenDialog component على ال form . وكما يفعل Delphi للقوائم ، فإنه يمثل ال dialog كايقونات غير مرئية عندما يكون البرنامج فى حالة تشغيل . يمكنك وضع ال MainMenu1 وال OpenFileDialog1 فى أى مكان تريده . يجب ان يشبه ما يعرض على شاشتك شكل (٢-١٠) ، والذي يوضح نافذة ال BitView form بعد التطوير فى Delphi .



شكل (٢-١٠) ، استخدام هذه الصورة لنافذة ال BitView form فى Delphi كمرشد لك عند تكوين ال application

٨- اختر ال OpenFileDialog1 component ، واضغط الزر البيضوى المجاور لقيمة خاصية ال Filter . هذا يفتح ال Filter Editor الخاص بـ Delphi ، والذي يمكنك استخدامه لتحديد الإمتدادات الخاصة بالملفات . على سبيل المثال ، يمكنك تحديد الإمتداد \* .bmp للملفات ، باستخدام الشكل (٢-١١) كمرشد . اضغط زر ال OK بال dialog لحفظ الإمتدادات . (بعد هذه الخطوات المرفقة ، سوف اصف طريقة أخرى للتعامل مع هذا ال string) .



شكل (١١-٢) : Filter Editor Filename الخاص بـ Delphi لتبسيط إدخال الـ filename filters

٩- جميع الـ components الآن في مكانها ، وقد تريد حفظ ، وتريد إجراء عمليتي الـ compile والـ link ومن ثم تشغيل البرنامج . اترك البرنامج لتعود لـ Delphi . (أوامر القائمة مازالت غير تامة ، ولا تستطيع استخدام أمر الـ FileExit لترك البرنامج ، لذا اضغط زر إغلاق البرنامج بدلاً من ذلك) .

١٠- في نافذة الـ form- ليس في البرنامج العامل- اختر FileOpen لإنشاء procedure لهذا الأمر . وفيما بين كلمتي البداية والنهاية الرئيسيتين للبرنامج ، أدخل البرمجة التالية (راجع الـ Bitview\Main.pas كمرشد ، أو افتح هذا الملف وانسخ عباراته) :

```
if OpenFileDialog1.Execute then
begin
    BitImage.Picture.LoadFromFile(OpenDialog1.FileName);
    Caption := OpenFileDialog1.FileName;
end;
```

١١- تقوم عبارة if السابقة بتنشيط الـ OpenFileDialog1 بإستدعاء object's Execute function . اذا كانت قيمة الـ function مساوية لـ True ، فإن المستخدم قد أغلق dialog بإختيار زر الـ OK الخاص به . في هذه الحالة ، ينفذ البرنامج العبارتين داخل كلمتي البداية والنهاية الرئيسيتين . وتستدعي العبارة الأولى LoadFromFile method للـ Picture object لتحميل الملف المسمى من قبل خاصية Filename الخاصة للـ dialog . وتحدد العبارة الثانية نفس اسم الملف هذا

للـ form's caption، مما يغير عنوان النافذة الى مسار bitmap. ورمز الـ (=): يعتبر عامل تخصيص من Pascal، والذي يخصص القيمة التي الى يمينه الى الـ object الذي الى يساره.

١٢- ارجع الى نافذة الـ form، اختر FileExit، أدخل عبارة Close; بين البداية والنهاية في الـ event-handler procedure، الذي ينشئه Delphi للأمر.

١٣- احفظ المشروع بالكامل، واضغط F9 لإجراء عمليتي compile، والـ link لتشغيل التطبيق. قم بتحميل أى ملف bitmap. اضغط زر الحجم المتوسط على الحد الأيمن العلوى للـ BitView لتقلص النافذة الى حجمها الطبيعي. أعد تحديد حجم النافذة بواسطة فأرتك، ولاحظ أن bitmap تنكمش أو تتسع لتلائم حجم النافذة. هذا قد يجعل بعض الصور تبدو غريبة، [أنظر المشروع (٢-٤) في نهاية هذا الباب لإقتراح تغيير هذه الخاصية.

لأن Image component قادراً على عرض أنواع متعددة من ملفات جرافيكية، لذا يسهل تحديث الـ BitView لعرض أيقونات، metafile وأنواع الصور الأخرى. ويمكنك أيضاً تحديد نوعية الـ standard file filter string ليتمكنك التعامل مع display حيث يعرض bmp و wmf و emf. وملفات أخرى ذات امتدادات جرافيكية معروفة.

ولعمل هذا التغيير، ارجع الى Delphi واستخدم الـ Object Inspector لإختيار الـ MainForm. اضغط Events page tab، واضغط مرتين داخل المساحة الخالية الى اليمين من الـ OnActivate event وهذا ينشئ blank procedure for the event، والذي يحدث عندما تتعامل مع الـ form (فى بداية البرنامج، على سبيل المثال). أو الإنتقال من الـ form الى الـ form. اجعل الـ procedure يبدو مثل هذا:

```
procedure TMainForm.FormActivate(Sender: TObject);
begin
  OpenFileDialog1.Filter := GraphicFilter(TGraphic);
end;
```

## الباب الثاني : معلومات حول الـ Visual Components

إن العبارة تحدد الـ GraphicFilter function التابعة للـ Graphics unit الى خاصية الـ OpenDialog1 Filter. وبدلاً من استخدام filter editor كما هو موضح من قبل، فإن هذا يقدم كل امتدادات اسماء ملفات الجرافيك لـ TGraphic class.

وتغيير آخر يمكنك أن تقوم به هو أن تضيف صور JPEG الى الملفات القياسية. لفعل هذا، حدد موضع uses declaration في بداية الـ source code، وأضف الـ JPEG unit الى الـ units الأخرى الموجودة في قوائم. ويجب أن تبدو uses declaration مثل هذا:

uses

Windows, SysUtils, Messages, Classes, Graphics, Controls,  
Forms, Dialogs, Menus, ExtCtrls, Jpeg;

والآن، عندما تقوم بتشغيل البرنامج (اضغط F9) فإن الـ Open dialog يضع ملفات الـ Jpeg و .jpg في قائمة مع أنواع ملفات الجرافيك المعروفة الأخرى.

### :System Components

تقدم System component مجموعة متنوعة من الـ objects للتوصل الى system hardware and software. إن الـ Components الموجودة في هذه الفئة تتضمن Timer for background processing، و PaintBox للجرافيك ذات الأغراض العامة، و MediaPlayer component للوسائط المتعددة والـ components الأخرى تتضمن الـ OLE components والـ DDE لتشارك البيانات.

في النهاية، أنك سوف تجد أمثلة لكل تلك الـ components في هذا الكتاب. في هذه الفقرة، سوف تستخدم Timer لإنشاء ساعة رقمية تعد ٢٤ ساعة، والتي تظهر أيضاً كيفية أداء الـ background processes.

### استخدام الـ Timer component:

إن الـ Timer component يعتبر واحداً من Delphi's simplest. فإن له خصائص قليلة و Single event يمكنك برمجته واحد أو أكثر لأداء مهام تتماشى مع تطبيقات أخرى، أو تعمل في الخلفية.

لإنشاء Timer component، اختر من فئة ال System component palette. أضف ال component object على ال form الخالية، قم بتغيير خاصية Name له اذا أردت ذلك، وقم بتحديد ال Interval الذى ترغبه لتكرار ال event حسب المدة الزمنية وتقاس بال milliseconds وهى من 1 الى 65535. إن قيمة ال 1000 تنشئ عداد لكل ثانية؛ وقيمة ال ١٠٠ تنشئ عداد لكل ١/١٠ ثانية. (ويسمح لك Delphi بتحديد فترة العداد بصفر، ولكن لا يوجد سبب معقول لهذا).

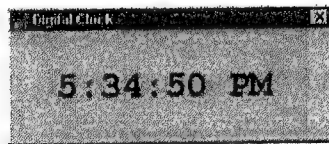
ول ال Timer component، event واحد، وهو ال OnTimer. اضغط مرتين قيمة ال event، أو يمكنك أن تضغط مرتين ال Timer object، والذى يقوم بنفس الشئ، وأدخل عبارات بين البداية والنهاية للتنفيذ حسب Timer's frequency.

### المثال: DClock:

لإظهار كيفية إنشاء Timer event handler، فإن هذه الفقرة تقدم مثلاً، وهو DClock. بالإضافة الى إظهار كيفية استخدام ال Timer component، فإن هذا البرنامج التعليمى يوضح كيفية إنشاء نافذة ذات حجم ثابت، والتى تعتبر مفيدة عندما لا يكون هناك فائدة فى السماح للمستخدمين بضبط حجم النافذة.

### تشغيل ال DClock:

يوضح شكل (٢-١٢) عرض لل DClock. وبخلاف ال MemoPad، فإن نافذة الساعة ليس لها أزرار تكبير وتصغير، وليس لها قائمة أو أوامر.



شكل (٢-١٢): عرض ال DClock يظهر background processing باستخدام Timer object

### إنشاء DClock:

افتح ملف ال DClock.dpr الآن وافحص component's form objects. افحص ال source code ال Main.pas للبرنامج فى القائمة (٢-٣). بعد

## الباب الثاني : معلومات حول الـ Visual Components

الإنهاء من الفحص ، أغلق التطبيق وحاول إنشاء الساعة الرقمية الخاصة بك بإتباع التعليمات الموجودة بعد القائمة .

القائمة (٢-٣) : Dclock\Main.pas

unit Main;

interface

uses

Windows, SysUtils, Messages, Classes, Graphics, Controls,  
Forms, Dialogs, ExtCtrls, StdCtrls;

type

TMainForm = class(TForm)

TimeLabel: TLabel;

Timer1: TTimer;

procedure Timer1Timer(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

MainForm: TMainForm;

implementation

{ \$R \*.DFM }



```
procedure TMainForm.Timer1Timer(Sender: TObject);
begin
    TimeLabel.Caption := TimeToStr(Time);
end;

end.
```

١- إبدأ مشروع جديد. قم بتغيير خاصية الـ Caption للـ form لتصبح Digital Clock وحدد الـ Name الخاص بها بـ MainForm. قم بإنشاء دليل مثل C:\Projects\Dclock لتخزين ملفات التطبيق، واحفظ المشروع. اجعل اسم ملف Main unit ولف المشروع Dclock.

٢- لإزالة أزرار التكبير والتصغير للنافذة، اختر الـ form (اضغط داخل نافذتها) واضغط مرتين خاصية أيقونات الـ BorderIcons. (ضع الفأرة على اسم الخاصية على اليسار، وليس على قيمتها على اليمين). لاحظ علامة الزائد الصغيرة على اليسار من اسم الخاصية، والتي تتحول إلى علامة ناقص بعد الضغط مرتين. علامة الزائد تخبرك أن هذه الخاصية لها واحد أو أكثر من القيم الفرعية المختبئة. علامة الناقص تشير إلى أن القيم الفرعية تم عرضها. في هذه الحالة، هناك ثلاث قيم من هذا النوع. حدد الـ biSystemMenu بـ True (القيمة الافتراضية له حسب نظام البرنامج)، والقيم الفرعية المتبقية بـ False. بالرغم من هذا التغيير، مازالت الـ form في Delphi توضح أزرار تكبير وتصغير. وهذا طبيعي لأنه يجب أن يمكن تغيير حجم النافذة عند تصميم التطبيق - يجب أن تقوم بتشغيل البرنامج لترى المظهر الحقيقي للنافذة، والذي هو في هذه الحالة، محدد بحجمه المصمم.

٣- وتلك السمة تتطلب تغييراً آخر - إزالة أزرار الحجم من النافذة وذلك لا يكفي لمنع المستخدمين من إعادة تحديد حجم إطار النافذة. لتثبيت حجم النافذة، اختر خاصية الـ BorderStyle الـ form، واختر bsSingle من قائمة اللائحة. لنفس السبب المذكور في الخطوة رقم (٢)، فسوف ترى تأثير هذا التغيير فقط عندما تقوم بتشغيل البرنامج.

## الباب الثاني : معلومات حول ال Visual Components

٤- أضف ال Standard Label component على ال form قم بتغيير خاصية ال Name لل object ليكون ال TimeLabel ، وقم بتغيير ال Caption الخاص به ليكون 00:00:00 AM وهذا يضمن أن ال label لها الحجم المناسب لأرقام الوقت المدخلة في وقت التشغيل .

٥- اختر خاصية ال Font لل TimeLabel . اضغط الزر البيضاوى لفتح حوار لإختيار ال font . اختر ال font ، حجماً ونمطاً لساعتك . إننى قد خصصت ال Bold Courier New مع حجم ٢٤ نقطة .

٦- أضف ال System Timer component على ال form ولأن هذا هو ال Timer object الوحيد للبرنامج ، فال Name الافتراضى وهو ، Timer1 ، يعتبر كاف . حدد الخاصية ال Interval لل Timer object بـ ١٠٠٠ ، والذي يساوى ثانية واحدة ، وهذه هى القيمة الافتراضية .

٧- اختر ال Timer1 object ، واضغط ال Events page tab فى ال Object Inspector . يجب أن ترى event وحيد وهو ال OnTimer . اضغط مرتين القيمة الموجودة الى اليمين من ال event لإنشاء ال procedure فى ال source code . أو ، يمكنك ببساطة أن تضغط مرتين ال Timer object ، الذى ينشئ برنامج ال event handler - وهو الوحيد فى هذا ال component . أدخل السطر التالى بين كلمتى البداية والنهاية الرئيسيتين :

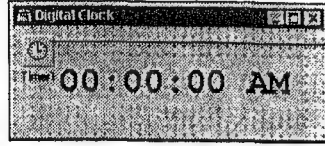
`TimeLabel.Caption := TimeToStr(Time);`

٨- والسطر السابق يحول الوقت الحالى الى ال string ويحدد النتيجة لل Caption الخاص بال label وهذا يحدد الوقت على الشاشة مرة كل ثانية- وهو الزمن الذى تم تحديده لل OnTimer event .

٩- قم بإعادة تحديد نافذة ال object لتلائم بشكل جيد . (استخدم ال View/Form أو اضغط F12 لتجد النافذة اذا كانت مخفية) . يجب أن يشبه شكل (٢-١٣) ، الذى تظهر ال DClock's form فى Delphi . وأيقونة ال Timer1 object لا تكون مرئية أثناء وقت التشغيل ، ويمكنك تحريكها الى أى مكان ملائم .

١٠- اضغط F9 لإجراء عمليتى ال compile وال link وتشغيل التطبيق . فى ثوان قليلة يجب أن ترى ساعة رقمية تدق الوقت . اضغط Alt+F4 أو استخدم أمر

ال System (أو اضغط زر الإغلاق فى ال Windows 95) للعودة الى  
. Delphi



This icon is not visible  
at runtime

شكل (٢-١٣): استخدم هذه الصورة للـ DClock's form  
فى Delphi كمرشد لك عندما تنشئ تطبيق

## :Win32 Components

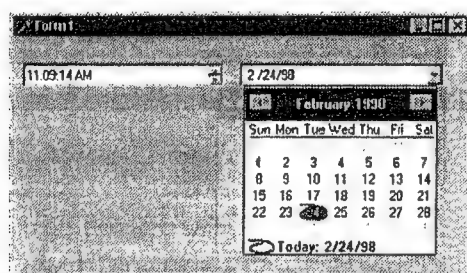
أنك من المحتمل أن تستخدم component أو أكثر من Win32 component فى كل تطبيق تكتبه. وهذه ال components تحتوى على native controls ذى 32-bit الخاص بتكوين أكثر من windows و dialog ، coodbars ، status bars ، animations .

وهذه الفقرة تقدم إحدى ال Win32 components ، وهو TDateTimePicker ، والذي يمكنك إضافته الى أى تطبيق يحتاج الى طريقة للمستخدمين لاختيار التاريخ والوقت. وسوف تتعامل مع Win32 components الأخرى عبر هذا الكتاب .

والآن ، فإننا نركز على ال TDateTimePicker إننى لا اعلم كم من الوقت أمضيت فى كتابة code للاستفسار من المستخدمين عن التاريخ والوقت ، ولكن مع component TDateTimePicker الخاص بـ Delphi ، لن احتاج الى إضاعة المزيد من الوقت فهذا ال component ليتح للمستخدمين ان يدخلوا ويختاروا التواريخ والمواقيت ولكنه يزيد عن مجرد المستفسر - فهو فى الواقع عبارة عن component يحتوى على نظام تاريخى كامل .

وال TDateTimePicker لا يمكن ان يكون ابسط استخداماً مما هو عليه . قم باسقاط إحدى هذه ال component على ال form ، وحدد خاصية ال Kind التابعة له بـ dtkDate لتحديد اليوم وال dtkTime لتحديد الوقت الحالى . لتعرف المزيد عن هذا ال component ، إتبع الخطوات التالية :

- ١- ابدأ تطبيق جديد .
  - ٢- اضغط Win32 page tab فى الـ Delphi's component palette .
  - ٣- اختر الـ DateTimePicker component ، اضغط على الـ form ، لتضيف الـ object عليه ، لإجعل خاصية الـ Kind مساوية للـ dtkTime .
  - ٤- اعد الخطوة رقم (٣) لإضافة DateTimePicker آخر على الـ form . حدد خاصية Kind بهذا الـ dtkDate (البديل الافتراضى) .
- يمكنك تشغيل البرنامج فى هذه النقطة بضغط F9 . يوضح الشكل (٢-١٤) عرض البرنامج و date component مفتوحاً (اختر قيمة للتاريخ بإختيارك لأحد القيم) .



شكل (٢-١٤)، TDateTimePicker يبسط اختيار قيم التاريخ والوقت

بعد تجربة الـ component ، قم بإنهاء البرنامج الاختيارى وعد الى Delphi . قم بتجربة مجموعة من القيم الخاصة بالخصائص التالية :

• **Checked:** اجعل الـ check box الى اليسار من الـ control's edit وعند الاختيار ، يتم التعامل مع الـ control ، وعند عدم الاختيار ، يتوقف التعامل مع الـ control يستطيع المستخدم إدخال أو اختيار التواريخ والمواقيت . وهذه الخاصية لا تمثل شيئاً الا اذا كانت خاصية الـ ShowCheckbox محددة بـ True .

• **Date:** التاريخ الذى يتم اختياره حالياً (والتاريخ حسب البديل الافتراضى هو هذا اليوم) مثل الـ TDate object استخدم هذه الخاصية لتحصل على تاريخ يتم اختياره من قبل المستخدم .

• **DateFormat:** اختر فيما بين Formats اما dfShort (٢٤ / ٢ / ٩٨) و dfLong (الثلاثاء ، ٢٤ فبراير ، ١٩٩٨) للعرض فى الـ control's edit وهذا القيم تؤثر فقط على الـ component ذات الـ Kind المحدد بـ dtkDate .

• **DateMode:** اختبر بين نمطين لـ Date controls ( Kind ) يساوى ( dtkDate ). استخدم dmComboBox (البديل الافتراضى للنظام) ليستطيع ان يختار منها المستخدم تاريخا استخدم dmUpDown لعرض اسهم الإشارة الى أعلى واسفل والتي، عندما تضغط، تقدم أو تؤخر الجزء المختار من الوقت. بصرحة، إننى اجد ان هذا الخيار الثانى يصعب استخدامه، بالإضافة الى ذلك.

• **Kind:** كما ذكرت، حدد هذا بـ dtkDate لضبط التاريخ أو dtkTime لإدخال التوقيت. والخصائص مثل الـ DateFormat والتي تحتوى على لفظ الـ Date فى اسمائها تكون ذات معنى للـ date controls فقط. رغم ذلك، حتى لضبط التوقيت، فإن مجال الـ Date لا يزال يقدم تاريخاً صالحاً (وهو اليوم، الا اذا كنت قد غيرته من خلال نظام التشغيل).

• **MaxDate, MinDate:** أدخل الحدين الأدنى والاقصى للتواريخ التى تريد ان يستطيع المستخدمين ادخالها فى control التاريخ ( Kind ) تساوى ( dtkDate ).

• **ShowCheckbox:** حدد هذه الخاصية بـ True لإظهار الـ check box الى اليسار من التاريخ أو التوقيت. وعند تشغيله (انظر خاصية الـ Checked)، فتصبح قادراً مع التعامل معه ويمكن تغييره. وعند إيقافه (يكون الـ checked مساوياً للـ false)، لا يستطيع المستخدمين تغيير قيمته. فكرة: استخدم خاصية الـ enable للتعامل مع الـ TDateTimePicker object أو إيقاف التعامل معه استخدم خاصيتى الـ Checked والـ ShowCheckbox اذا أردت أن تجعل المستخدمين قادرين على التعامل أو إيقاف التعامل مع الـ control.

• **Time:** الوقت الذى يتم اختياره حالياً (الآن هو الوقت حسب البديل الافتراضى)، مثل الـ TTime object. استخدم هذه الخاصية للحصول على وقت يتم اختياره من قبل المستخدم.

**فكرة:** قم بإبراز وحذف المدخل الافتراضى حسب النظام فى خاصية Time. هذا يؤدي الى عرض 12:00:00 A.M. إن حذف خاصية الـ Date يحددها بـ ١٢ / ٣٠ / ٨٩. وهو السبت، ٣٠ ديسمبر، ١٩٨٩.

## الباب الثاني : معلومات حول الـ Visual Components

إذا كان برنامجك يحتاج الى تنبيه الى أن المستخدم قد إختار تاريخ جديد فقم بإنشاء الـ event handler للاستجابة عندما يتغير التاريخ المختار . هناك طريقتين لعمل هذا . اختر الـ TDateTimePicker object ، واضغط الـ Events page tab ، فى الـ Object Inspector . اضغط مرتين الـ OnChange والـ OnCloseUp ، وأدخل الـ code لكلاهما (أو يمكنك ان تكتب procedure منفصل وتستدعيه من خلال الـ event handler) :

```
var
  S: String;
begin
  S := DateToStr(DateTimePicker1.Date);
  ShowMessage('Date entered = ' + S);
end;
```

ويقوم الـ procedure باستدعاء DateToStr (و function فى SysUtils unit) لتحويل الـ Date الى string ، والذي يتم تمريره للـ ShowMessage لعرض التاريخ الذى تم اختياره فى نافذة dialog box من الواضح ضرورة برمجة الـ OnChange والـ OnCloseUp حتى ان التغييرات اليدوية لقيم التاريخ وكذا والتواريخ المتقاء من الـ object يتم تسجيلها .

### افكار للمستخدم الخبير

● فى غالبية الحالات ، بعد إضافة الـ component على الـ form يجب ان تعطى للـ object اسماً يعكس هدفه . والإستثناء المعتاد لهذه القاعدة هو عندما تدخل نوع واحد من الـ component object ، على سبيل المثال ، اذا كان برنامجك له قائمة رئيسية واحدة ، فإن الاسم الافتراضى حسب النظام - MainMenu1 - يكون كافياً .

● استخدم اسماء واضحة مثل CloseButton ، و AgeLabel ، و NameEdit . بعض واضعى البرامج يودون استخدام اسماء قصيرة . على سبيل المثال B1 ، B2 ، B3 للازرار ولكن اذا فعلت هذا ، قد لا تعرف على الـ objects

فى العام المقبل أو الشهر التالى - أو، اذا كانت ذاكرتك ليست افضل من ذاكرتى،  
غداً فى المساء .

● كقاعدة بديهية، ان برامجك سوف تصبح اسهل فهماً وحفظاً اذا الحقت  
اسماء ال component بنهاية قيم خاصية ال Name. على سبيل المثال، قد يكون  
اسم ال NoteMemo افضل للـ Memo من اسم ال NoteMemo. فالاسم الاول  
يذكرك بنوع component object؛ والآخر يخفى هذه الحقيقة المهمة .

● يجب ان تبدأ الاسماء بحروف أو وضع خط تحت الكلمة، وقد تحتوى  
على حروف وارقام وخطوط تحت الكلام فقط . ويعتبر Label10 اسم صحيح،  
بينما Label10 ليس كذلك . لا يجب ان يكون فى ال Names مسافات فيما بينها .  
استخدم حروف الكبيرة لتوضيح الاسماء الطويلة - مثلاً، يعتبر ال  
MainSysMenu مقروءاً بشكل اوضح من ال mainsysmenu. قد يحتوى  
الاسم على خطوط فيما بينه كما هو فى My\_Main\_Menu. ان الخطوط يصعب  
رؤيتها على الشاشة، رغم ذلك، ولا احبذ استخدامها. وقد تتكون الاسماء من  
واحد الى ٦٣ رمز ويجب ان تلتزم بتركيب الجمل المعرفة فى لغة Pascal.

● وبخلاف ال Names، فإن ال Captions وخصائص قيم النص الاخرى  
قد يكون لها أية رموز يمكنك كتابتها أو قصها أو لصقها، فى أى ترتيب، بما فى ذلك  
المسافات وعلامات الترقيم. وطول ال Caption محدد بـ ٢٥٥ رمز، ولكن من  
الناحية العملية، فإن حجم ال objects يحدد طول ال Caption الخاص به الى أقل  
من هذا بكثير .

● عندما تصبح ال form ممثلة بال component object، أو عندما يغطى  
object واحد أو اكثر مساحة ال form، فقد تواجه مشكلة فى اختيار ال form فى  
هذه الحالة، استخدم قائمة ال Object Inspector لاختيار ال form بالاسم .  
على سبيل المثال، لاختيار ال form الرئيسية فى معظم تطبيقات هذا الكتاب، اختر  
MainForm من قائمة ال Object Inspector. يمكنك استخدام هذا الاسلوب  
لاختيار ال component object تم إضافته على ال form.

● لمحاذاة components متعددة، أولاً قم بتحريك component واحد الى  
وضعه الصحيح. اختر هذا ال component أولاً ثم اختر الآخرين. ويقوم أمر

## الباب الثاني : معلومات حول الـ Visual Components

الـ Edit|Align بمحاذاة الـ component object بالنسبة للـ component الذى اخترته أولاً.

● الـ Menu Designer ، اضغط Ctrl+Ins لإدخال أمر قائمة بين أمرين آخرين اضغط Ctrl+Del لحذف أمر . اكتب شرطة واحدة فى خاصية Caption لأمرها لإدخال فاصل بين أمرين آخرين .

● اختر Edit|Lock Controls لتشغيل خيار هذه القائمة (تظهر علامة صح . بالقرب منه) . عند تشغيله ، تمنع هذا الخيار تحريك أو إعادة تحديد حجم الـ component object ، ولكن لا يزال بإمكانك تعديل events وخصائص الـ control . بعد ان تقوم ببرمجة الـ form بالطريقة التى تريدها ، استخدم هذا الأمر لمنع إعادة الترتيب الذى قد يحدث للـ objects الموضوعه بعناية عن طريق الصدفة .

● اذا لم تتمكن من تحريك أو إعادة تحديد حجم الـ component object المختارة ، تنظر اذا ما كانت الـ Lock Controls المذكور فى الفكرة السابقة عاملة .

● والمستخدمين الجدد للـ Pascal دائماً ما يقلقون بشأن وضع الفصلة المنقوطة ، ولكن لا تبالغون فى إهتمامكم . اذا قمت بكتابة فصلة منقوطة فى مكان خطأ ، يقوم Delphi بكتابة رسالة Error فى الـ C والـ C++ ، تنهى الفصلة المنقوطة العبارات . فى Pascal ، تقوم الفصلة المنقوطة بفصل العبارات عن بعضها البعض . على سبيل المثال ، يمكنك الا تضع فصلة منقوطة قبل جزئية else فى عبارة if لأنه ، وعلى عكس الـ C والـ C++ ، الـ if-then-else الخاصة بـ Pascal تعتبر عبارة واحدة . إرجع الى قوائم هذا الكتاب للحصول على امثلة المواضيع الفصلة المنقوطة الصحيحة . يمكنك ان تقضى بعض الحظاظ فى الاختبار مع code editor's code insights التى كما هو مفصل فى الباب الأول ، تقوم بادخال عبارات Pascal فى تركيب الجمل الصحيح تلقائياً .

● ولإضافة أكثر من الـ component objects من نفس النوع بطريقة اسرع ، اضغط مرتين الـ component icon فى الـ Delphi's palette . وهذا يؤدي لإضافة الـ object لهذا الـ component فى منتصف الـ form استخدم هذا الاسلوب لإضافة الـ component object ومتعددة - مجموعة من الـ check



boxes، مثلاً. وكل object يقع فوق الآخرين السابقين تماماً، ويجب ان تضغطهم وتسحبهم الى اماكنهم النهائية قبل ان تفعل شئ آخر.

• واسرع طريقة لإضافة ال component object ومتعددة ومن نفس النوع هى ان تدخل واحداً، وتدعه مختاراً، ثم تضغط Ctrl+C يتبعها Ctrl+V لأى عدد من ال objects ترغبه. وتعطى ال object اسماء مرقمة متتالية مثل Label1، Label2، و Label3.

• لإضافة component object على ال form، اضغط مرتين على ال object لإنشاء ال event handler الخاص به. وهو دائماً أول event موجود فى نافذة ال Object Inspector. بعض ال component لا تعمل بهذا الشكل. فمثل، ان الضغط على ال MainMenu مرتين تفتح نافذة ال Menu Designer.

• لكى تجد ال event handler متواجداً، اضغطه مرتين ال Events page بنافذة ال Object Inspector. هذا يعرض ال form's unit، ويضع المؤشر داخل event handler، ويقوم ال Delphi بإنشاء event handler جديد فقط اذا لم يكن هناك واحداً موجود بالفعل.

• حدد خاصية ال AutoSize لل Label بـ True لتحصل على التحكم بتعديل حجمها تلقائياً لتلائم بيانات ال Caption الخاصة بها. حدد هذه الخاصية بـ False اذا كنت تريد أن يظل ال component ذا حجم ثابت فى كل الأوقات يجب أن يستخدم حجماً ثابتاً اذا حددت ال WordWrap بـ True لعرض ال labels متعددة السطور.

• إن طول ال captions لنافذة أو طول single-line string تجد من الصعوبة عرضها لأن حجم ال Object Inspector صغير ولعرض بصورة أسهل، أدخل النص فى أى مكان من نافذة code editor قم بإبراز النص، اضغط Ctrl+X لقصه الى ال Windows clipboard، إرجع الى ال Object Inspector، قم بإبراز خاصية نص مثل ال Caption، اضغط Ctrl+V لإدخال نصك. إننى أود الاحتفاظ بمثل هذه ال Strings العديدة كتعليقات فى ملفات ال source code الخاصة بى، فيما بين قوسى تعليقات (\* and \*). يمكننى، عندئذ، عرض النص بسهولة ونسخه (اضغط Ctrl+C بدلاً من Ctrl+X) من ال clipboard الى ال

## الباب الثاني : معلومات حول الـ Visual Components

Caption . على سبيل المثال ، أدخل تعليقات مثل التالي في أى مكان في unit's source code

```
(*  
This is the first window caption  
This is another window caption  
*)
```

• اذا كنت تريد أن تقوم بنفسك بتحليل مداخل الوقت والتاريخ في TDateTimePicker لتقييد المداخل بتواريخ معينة ، مثلاً ، الأيام من الاثنين الى الجمعة - فقم بتحديد خاصية الـ ParseInput بـ True . وهذا ينفذ الـ OnUserInput event عندما يكتب المستخدم في نافذة الـ edit .

### المشروعات التي يمكنك تجربتها

(١-٢) : قم بتغيير خصائص متنوعة في الـ component object لـ MemoPad . على سبيل المثال ، عين Font مختلف لـ Memo1 object . قم بتغيير الـ MemoLabel ، وعليك تجربة خاصية الـ Color التابعة له .

(٢-٢) : عندما تقوم ببدء تطبيق الـ DClock ، فإنه يعرض أولاً 00:00:00 AM قبل إظهار الوقت الحالي . لتحسين العرض وإظهار الوقت بمجرد ظهور نافذة البرنامج ، قم بإنشاء الـ OnCreate الخاص بالـ form . اضغط خلفية الـ form لإختيار الـ form ، ثم اختر الـ page Events tab في الـ Object Inspector . اضغط السهم المشير لأسفل الى اليمين من الـ OnCreate ، واختر الـ Timer1Timer . هذا هو نفس الـ event handler الذى قمت ببرمجته لـ Timer1 . عندما تقوم بتشغيل البرنامج ، فإنه يبدأ الـ TimeLabel بإستدعاء الـ label بالوقت الحالي قبل أن تصبح النافذة مرئية .

(٣-٢) : افتح مشروع الـ BitView ، واختر الـ OpenFileDialog الخاص به . إن خاصية الـ Ctrl3D متوفرة للملاحظة الخلفية فقط ، وفى الـ

Windows 95 و NT، لم يعد لها أى تأثير. إن جميع ال controls فى هذه الانظمة العاملة لها خاصية 3D رغم ذلك، اذا كنت لا تزال تستخدم نسخة اولية من Delphi أو Windows 3.1، نحاول تغيير خاصية ال Ctrl3D ل False، واضغط F9. عندما تختار امر البرنامج FileOpen، فإنك ترى Windows dialog box قياس، وغالباً ذا خلفية بيضاء ومظهر واضح نسبياً. للعودة الى تأثير ال 3D، قم بتغيير خاصية ال Ctrl3D مرة اخرى الى True.

(٢-٤): افتح مشروع ال BitView واختر ال BitImage. حدد خاصية ال Stretch للصورة ب False، واضغط F9 افتح ملف bitmap إنه يعرض بشكل اسرع لان تعديل احجام الصور يستغرق وقتاً. رغم ذلك، لسوء الحظ، ان عرض صور ذات الحجم الكامل قد يقتطع المساحات التى تمتد بعد حدود النافذة. فى الباب القادم، سوف تدخل scroll bars فى النافذة لحل هذه المشكلة. ايضا راجع الباب العاشر لمزيد من المعلومات حول scrolling.

## المخلص:

• ان Visual component تبسط عملية برمجة ال Windows. لإنشاء تطبيق Delphi، قم بإضافة ال component object على ال form، واختر ال event وخصائص ال objects.

• اختر فيما بين فئات ال Delphi's component باختيار واحداً من page tab الموجودة فوق ال component palette. اضغط component icon الذى تريده، قم بتحريك مؤشر الفأرة داخل ال form، واضغط مرة اخرى لإضافة ال component فى هذا الموضع.

• ان اختيار component يؤدي الى إحاطته بمربعات سوداء والتى يمكنك استخدامها لاعادة تحديد حجم ال object. يمكنك أيضاً ضغط وسحب ال component لوضعها على ال form اختر component بضغطه مرة واحدة، أو اختره بالاسم من قائمة اللائحة فى ال Object Inspector. يمكنك أيضاً اختيار components متعددة بضغط وسحب ال outline الموجود حولها.

## الباب الثانى : معلومات حول الـ Visual Components

● ان الخصائص هي قيم تحدد سمات الـ components على سبيل المثال ، Buttons و Labels لها خاصية Font لاختيار اسم الـ form ، وحجم النقطة والنمط .

● تمثل الـ event افعال اثناء وقت التشغيل خاصة بالـ components . ان ضغط قيمة الـ event مرتين يؤدي الى إضافة الـ event-handler فى الـ source code لبرمجة الـ event إضف unit أو أكثر من عبارات Pascal بين كلمتى البداية والنهاية الرئيسيتين والتان موضوعتان تلقائياً بواسطة Delphi .

● عندما تختار أكثر من components ، تعرض نافذة الـ Object Inspector الـ event المشتركة والخصائص المشتركة فيما بينهما . وتؤثر أية تغييرات تفعلها فى هذه القيم على كل الـ components المختارة .

● يحافظ الـ Delphi على التمثيل البصرى لتطبيقك فى الـ form وتمثيل الـ source code للـ Object Pascal فى نافذة الـ unit ، كلاً التمثيلين يتطابقان بحيث ان التغييرات فى إحدى النافذتين تنعكس فى الاخرى .

● تشير علامة زائد صغيرة الى اليسار من الخاصية الى ان هذه الخاصية لها قيم فرعية . اضغط مرتين حقل قيم الخاصية لوضع هذه القيم فى قائمة .

● ويشير الزر البيضاوى الى اليمين من قيمة الخاصية الى ان ضغط القيمة مرتين ، أو ضغط الزر مرة واحدة يفتح dialog boxes .

لتغيير موضع هذه الخاصية وبعض الخواص لها كلاً من القيم الفرعية و dialog boxes .

● استخدم الـ TDateTimePicker component من لوحة الـ Win32 لإنشاء control يستطيع من خلاله المستخدمون ادخال أو اختيار التواريخ والوقت . فى الباب التالى ، سوف تعرف المزيد عن الـ form والخصائص و الـ event . سوف تعلم ايضاً كيفية الاستجابة لاعمال الفأرة كيفية إضافة الـ scroll bars للـ form وكيفية ان تجعل النافذة مستقرة على القمة ، وكيفية إنشاء splash screen .

\*\*\*\*\*



# الباب الثالث

## معلومات أولية حول الـ Forms

### محتويات الباب:

Forms as components •

Form templates •

• استخدام dialog النافذة الرئيسية

• إغلاق نافذة

Form frameworks •

Data modules •

The splitter component •

إن كلمة الـ form، قد تصف أشياء كثيرة. في Delphi كل مشروع له على الأقل form واحدة تمثل النافذة الرئيسية للبرنامج. كما يمكن أن يكون للمشروعات أيضاً form متعددة للنوافذ الصغيرة، dialog box، وشاشات لإدخال البيانات. بالرغم من أن الهدف الرئيسي للـ form هو حمل components أخرى مثل الأزرار، check boxes، ويمكن للـ form أيضاً أن يؤدي أعمال استجابة للـ event، مثل الضغط على لوحة المفاتيح والفأرة.

والـ forms تعتبر ضمن versatile objects الأكثر تنوعاً واستخدامها بكثرة لتطوير التطبيقات. في هذا الباب، سوف تعرف الخصائص الرئيسية للـ forms وخواصها والـ events.

## :Forms as Components

تمثل ال form المظهر البصري لناظرة، ولكن ال form ليس مجرد ناظرة أخرى جميلة. إن ال form فى الحقيقة عبارة عن Delphi component لها مجموعتها الخاصة من ال events والخصائص. وبخلاف ال events الأخرى، رغم ذلك، لا تظهر ال form فى component palette.

من الطبيعى أنك تنشئ ال form بواحدة من إحدى الطريقتين: ببدء تطبيق جديد أو بإختيار أمر ال FileNew Form. استخدم المنهج الأول لإنشاء ال forms الرئيسية لتطبيقك. واستخدم المنهج الثانى للحصول على forms إضافية- لإنشاء About dialog، مثلاً، أو لعرض شاشة بدء تشغيل واسعة. (المزيد عن هذه الموضوعات سيأتى لاحقاً).

## :Forms and units

عندما تقوم بإنشاء forms جديدة، يضيف Delphi نافذتين لبيئة البرمجة. تظهر ناظرة ال form التمثيل البصرى لل form كما تظهر فى التطبيق التام. وتضع ناظرة ال code editor قائمة ببرمجة ال Object Pascal لل form. والنص الموجود فى هذه النافذة يسمى unit. يوضح شكل (٣-١) ناظرة ال code editor وناظرة ال form لتطبيق ال DClock من الباب الثانى. وداخلياً، ال form عبارة عن object unit، والتي تسمى فى بعض الأحيان module.

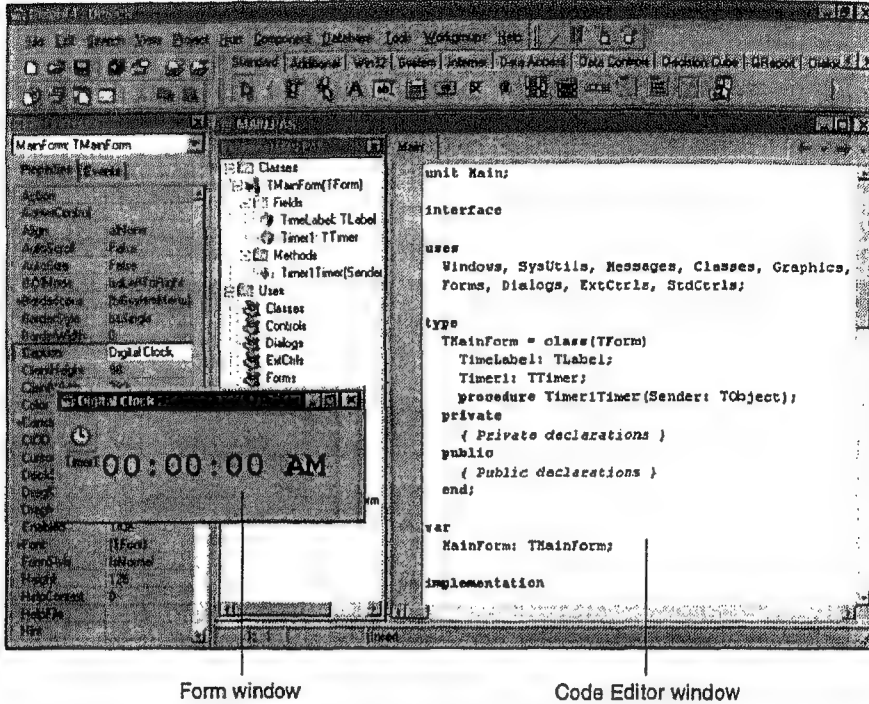
يقوم Delphi تلقائياً بإنشاء Object Pascal unit لكل forms جديدة تقوم بإدخال ما فى التطبيق. ليس من الممكن أو المستحب إضافة أكثر من form فى unit واحدة. يجب أن يكون لكل unit، form تقابلها. تقدم بعض ال unit بيانات و code ل modules أخرى لتستخدمها.

**تحذير:** الى واضعى البرامج الخبراء فى Pascal، كونوا على حذر! إن محاولة أكثر من form فى unit يمكن أن يعوق البرمجة البصرية والإمكانات لإنتاج ال code التابعة ل Delphi. للحصول على أفضل النتائج، إلزم بقاعدة form واحدة ل unit واحدة. إن Delphi لا يضيف أكثر من form فى ال unit الواحدة، فإن الطريقة الوحيدة التى تستطيع أكتساب الخبرة



### الباب الثالث: معلومات أولية حول الـ Forms

من هذا الخطأ هي بكتابة units جديدة من لاشئ، أو بمحاولة جمع أثنان من الـ form units في واحدة. فلا تفعل هذا.



شكل (٣-١)، Delphi يحتفظ بنافذة الـ form، تحمل Digital Clock labeled والبرمجة المرتبطة بها في نافذة الـ code editor (التي تحمل عنوان Main.pas)

### حفظ الـ forms في مشروعات:

عند حفظ مشروع جديد، لا تقم بتحديد خاصية Name للـ form وإسم ملف الـ unit بنفس الاسم. على سبيل المثال، إذا حددت خاصية الـ Name للـ form بـ MainForm، لا تقم بحفظ الـ form unit على أنها Mainform.pas (إنني اقترح استخدام Main.pas كبديل). لأن الأسم الداخلي للـ unit يساوي إسم الملف الخاص بها ناقص الامتداد .pas.، فإن استخدام نفس الاسم للـ form والـ unit ينتج عنه تضارب في الاسم.

لتجربة هذا الخطأ حتى تتلافاه في المستقبل، إتبع هذه الخطوات:

١- ابدأ تطبيق جديد وقم بتغيير خاصية الـ Name للـ form لتصبح Main.



٢- احفظ المشروع، وادخل Main كأسم ملف الـ unit. يقوم Delphi تلقائياً بإضافة الامتداد pas. لإنشاء اسم الملف الفعلي، Main.pas.

٣- اضغط زر OK الخاص بـ file-save dialog أو اضغط Enter. بسبب تضارب الاسماء، بدلاً من حفظ الملف، يعرض Delphi رسالة خطأ، " البرنامج يحتوى بالفعل على form أو module تسمى Main ". اضغط OK لإغلاق نافذة رسالة الخطأ.

٤- تحل المشكلة بتغيير خاصية الـ Name للـ form لاسم يختلف عن اسم الملف الخاص بها- على سبيل المثال، MainForm. يمكنك الآن ان تعيد الخطوة رقم (٢) وان تحفظ الوحدة كـ Main.pas.

لمنع هذا النوع من تضارب الاسماء، والذي قد يتكرر حتى تعتاد على قواعد التسمية في Delphi، إتبع الأفكار التالية لاختيار خصائص الـ Name واسماء ملفات الـ unit:

- أضف كلمة Form لكل Form; name property والأمثلة على اسماء الـ form الجيدة هي MainForm للنافذة الرئيسية للبرنامج، AboutForm للـ About dialog، و EntryForm لشاشة إدخال البيانات.

- احفظ الـ unit modules مستخدماً الـ Name التابعة لهم ناقص كلمة Form على سبيل المثال، عندما تقوم بحفظ المشروع، حدد اسم الملف Main للـ MainForm module، و About للـ AboutForm، و EntryForm للـ EntryForm. وهذا يؤدي الى إنشاء الملفات الثلاثة Main.pas و About.pas و Entry.pas.

### خصائص الـ form المختارة:

تعرض بعض النماذج من خلال هذا الكتاب كثير من الخصائص للـ form object. الكثير من الخصائص لها وظائف معروفة، ولذا لن اشرحها مرة أخرى هنا. (فخصائص الـ Height والـ Width للـ form لا تحتاج لشرح وظائفها). وكذلك إنك قد قابلت كثير من خصائص الـ form مثل الـ Caption والـ Name والـ BorderStyle. وفيما يلي اقوم بشرح خصائص الـ form-object الأكثر غموضاً والتي يمكنك رؤيتها وتحريرها في نافذة الـ Object Inspector.

## الباب الثالث : معلومات أولية حول الـ Forms

**ملحوظة:** كن على حذر من ان بعض الخصائص قد ورثتها الـ TForm من class أخرى مثل TComponent وبشكل عام، هذا ينطبق على غالبية class component اذا بحثت عن class فى وثائقك المطبوعة، فإنك تحتاج ان تفحص التسلسل الأعلى لتجد كل الخصائص والـ class methods.

Note

● **ActiveControl:** حدد هذه لأي component object مثل الـ Button أو CheckBox، باعطاء الـ object الـ input Focus عند بدء ظهور الـ form، ان عملية الـ input Focus ليس بالشئ الثابت، فيمكن للمستخدمين أن يضغطوا Tab لتحريك الـ input Focus من component لآخر.

● **AutoScroll:** حدد بـ True لتظهر وتختفى الـ scroll bars تلقائياً فى نافذة حتى لو ان هناك مزيداً من المعلومات للتعامل مع الـ window borders ان هذه هى القيمة الافتراضية حسب النظام، والتي تؤثر ايضاً على الـ Form Scaling اذا كانت الـ AutoScroll محددة بـ False، فإن كلاً من مساحة الـ client area ومستطيل النافذة قد تم تغيير مقياسهم، واذا كانت True، فإن الـ client area للـ form فقط هى التى يتم تغيير قياسها وغالباً مع الـ Scaled form التى يتم تغيير قياسها، يجب عليك ان تحدد الـ AutoScroll بـ False. لمزيد من المعلومات حول استخدام الـ scroll bars انظر الـ "Form Frameworks" فى هذا الباب.

● **Cursor:** يعين شكل المؤشر للعرض عندما يتحرك مؤشر الفأرة فى الـ client area للنافذة اختر إحدى القيم من القائمة لهذه الخاصية.

● **Enabled:** عادة تكون محددة بـ True لكى تستجيب الـ form للفأرة أو الـ timer، أو الـ Key Board event حددها بـ False لإيقاف الـ event handling لا تفعل هذا للـ form. حدد هذه الخاصية بـ False فقط للـ form التى يتحكم فيها البرنامج. (ان اسلوب الـ splash-screen الموضع فى هذا الباب يستخدم هذه الخاصية).

● **HorzScrollBar:** حدد القيم للـ scroll bar الأفقى فيظهر الـ scroll bar اذا كانت القيمة الفرعية Visible قيمتها بـ True وكان الـ Range اكبر من الـ

**ClientWidth**. اضغط مرتين اسم الخاصية (ليس قيمتها) لفتح القيم الفرعية هذه.

● **Icon**: تختار ايقونة لتمثل ال form اذا كانت ال form هى النافذة الرئيسية للبرنامج، تحدد خاصية ال Icon أيقونة النظام التى يعرضها ال Windows عندما تقوم بتصغير النافذة.

● **KeyPreview**: تحدد بـ True اذا كنت تريد ان تتلقى ال form اغلب ال keyboard event قبل الانتقال إلى أى Focused control اذا كانت False، فإن كل ال keyboard event تذهب للcontrol المختار بغض النظر عند تحديد ال KeyPreview، ان ال form لا تتلقى ابدأ ال keyboard event كالتى تحدث نتيجة الضغط على Tab، وال BackTab، ومفاتيح الأسهم، الا اذا كان ال control الحالى (مثل input box، مثلاً) يتعامل مع هذه المفاتيح.

● **Menu**: على وجه الخصوص ال MainMenu component يستخدم ك menu bar لل MainMenu الاول التابع لل form، ولكن يمكنك تغييره لقائمة اخرى. انظر الباب الخامس لمزيد من المعلومات.

● **ObjectMenuItem**: يستخدم مع تطبيقات ال OLE. لمزيد من المعلومات انظر الباب السادس عشر.

● **PixelsPerInch**: تحدد كيف يقوم التطبيق بإنشاء ال form ليتم قياسها بالنقاط لكل بوصة. استخدمه مع Scaled لإنشاء form لتبدو متشابهة فى الحجم فى Screen Resolution مختلفة. اذا كان ال Scaled محدد بـ False، فإن ال PixelsPerInch لا يكون لها أى تأثير. واسم هذه الخاصية الى حد ما مضلل لان قيمته لا تساوى عدد النقاط فى البوصة الواحدة. وكبديل، فإن Delphi يحدد ال PixelsPerInch على أساس حجم font النظام، ثم يستخدم هذه القيمة لقياس ال form.

● **Position**: تحدد ال method لحساب حجم ومكان ال form والتحديد الافتراضى حسب النظام، وهو poDefault، يعرض النافذة فى حجمها وموضعها

### الباب الثالث : معلومات أولية حول الـ Forms

المصمم ، فإن الـ poDefaultPosOnly يحدد المكان للنافذة كما هو مصمم ، ولكن بحسب حجمها في وقت التشغيل ، (يحدد الـ Windows الحجم المبدئي للـ form في وقت التشغيل) ؛ أما الـ poDefaultSizeOnly تحدد حجم النافذة كما هو مصمم ؛ ولكن الموقع يحدد اثناء وقت التشغيل (يحدد الـ Windows اين تظهر النافذة) ؛ والـ poScreenCenter يجعل النافذة في الوسط اثناء العرض .

● **Scaled** : تحدد بـ True لاستخدام خاصية الـ PixelsPerInch وانشاء الـ form التي يتم تغيير مقاساتها تلقائياً بالنسبة لحجم الـ form هذا يساعدك على إنشاء تطبيقات تبدو جيدة بغض النظر عن موقع العرض . تحدد بـ False لعدم تغيير مقاسات الـ form هام جداً لضمان ان النص يعرض بشكل سليم في الـ Form's control على انظمة تستخدم fonts كبيرة . وهذا يعتبر هام للغاية في الـ Windows 98 والـ NT 5.0 ، والتي يستطيع المستخدمون اختيار قياسات الـ form عشوائياً .

● **Tag** : لا يوجد تحديد مسبق . استخدم الـ Tag لتخزين أى قيمة عدد صحيح تريده - مثلاً ، رقم نسخة ، أو code أو عدد يفحصه البرنامج اثناء وقت التشغيل .

● **VertScrollBar** : تماماً مثل الـ HorzScrollBar ، ولكن يشكل الخصائص الفرعية للـ scroll bar الرأسى للـ form .

● **Visible** : تحدد بـ True ليجعل الـ component مرئياً ؟ أو False ليخفيه حتى يستدعى البرنامج Show method للـ component object . خاصية الـ Visible للـ form عادة تكون لان التطبيقات تجعل الـ form الخاصة بها مرئية تلقائياً في الوقت المناسب وللـ form المستخدمة نافذة صغيرة ، dialog box ، يمكنك تحديد الـ Visible بـ False لإخفاء النافذة لحين الحاجة .

● **WindowMenu** : في تطبيقات الـ Multiple Document Interface (MDI) ، تصمم القائمة لعرض عناوين للنوافذ المفتوحة . حدد هذه الخاصية لأى بند قائمة (عادة تلك التي تحمل عنوان "Window") في شكل الـ MainMenu للـ form .

**ملحوظة:** انظر الباب الخامس عشر، تطوير تطبيقات ال MDI، لمزيد من المعلومات حول تطبيقات ال MDI واستخدام ال form لإنشاء ال child windows.

Note

### ال form events المختارة:

كما هو الحال في خصائص ال form، فإن النماذج الموجودة في هذا الكتاب تظهر الكثير من ال events، والبعض الآخر - مثل OnClick وال OnDbClick- تكون واضحة فيما يلي ال event الأقل وضوحاً المعروضة في نافذة ال Object Inspector لل form المختارة:

● **OnActivate:** يتم استدعائه عندما يقوم البرنامج بالتعامل مع أى form، وعندما تتلقى لأول مرة input focus مثلاً عندما تنتقل الى البرنامج من تطبيق آخر. انظر أيضاً ال OnDeactivate.

● **OnClose:** يتم استدعائه عندما تغلق ال form، انظر أيضاً ال OnDestroy.

● **OnCloseQuery:** يستدعى قبل ان تغلق ال form مباشرة، غالباً عندما يستدعى التطبيق ال Close procedure. يمكنك استخدام هذا ال event لمنع فقد البيانات سؤال المستخدمين عن موافقتهم على حفظ الملفات التى تم تغييرها قبل ان ينتهى التطبيق، أو لمنع form من ان تغلق. لمزيد من المعلومات، انظر "إغلاق نافذة" فى هذا الباب.

● **OnCreate:** يستدعى مرة واحدة عندما يقوم البرنامج بإنشاء ال form object فى الذاكرة. استخدم هذا ال event لتنفيذ البدء مرة واحدة.

● **OnDeactivate:** يستدعى عندما يتحول المستخدم عن التطبيق. انظر ال OnActivate أيضاً.

● **OnDestroy:** يتم استدعائه قبل تدمير ال form object استخدم هذا ال event لإنهاء تواجد أى resource يمتلكها النظام اذا كانت ال form هى نافذة البرنامج الرئيسية، فإن ال OnDestroy يعتبر فرصتك الأخيرة لأداء أى عمل قبل إنتهاء البرنامج.

### الباب الثالث : معلومات أولية حول الـ Forms

● **OnDonner, OnBlitzen** : تحدث عند حدوث أى event مهما كانت

للـ form .

● **ODockDrop, OnDockOver** : يحدث الـ event إستجابة لحدوث

أسلوب docking أما بالنسبة للـ events الذى لهم علاقة بهما . مثل الـ OnEndDock ، والـ OnStartDock ، والـ OnUndock . لمزيد من المعلومات ، انظر "إنشاء الـ Docking Controls" فى الباب الثانى عشر .

● **OnDragDrop, OnDragOver** : جزء من أسلوب

drag-and-drop الخاص بـ Delphi . لمزيد من المعلومات عن انتقال البيانات ، والـ DDE والـ OLE ، انظر الباب السادس عشر ، أسلوب التعامل مع Clipboard ، والـ DDE والـ OLE .

● **OnHide** : يستخدم لأداء اعمال عندما تكون الـ form مختفية . فمثلاً ،

يستطيع الـ OnHide محو الذاكرة أو أى resources أخرى عندما تكون الـ form فى حالة not visible .

● **OnKeyDown** : يستدعى عندما يضغط المستخدم أى مفتاح ، بما فى ذلك

مفاتيح التى لها وظيفة المفاتيح الخاصة . استخدم هذا الـ event فى حالة التعامل مع أكثر مع مفاتيح فى نفس الوقت ، مثل ضغط مفاتيح الـ Alt والـ Shift ، مع مفتاح أخرى يمكن للبرنامج ان يتلقى أكثر من OnKeyDown events دون حدوث الـ OnKeyUp events فيما بين ذلك . قم دائماً بتحديد الـ KeyPreview بـ True عند استخدام هذا الـ event .

● **OnKeyPress** : تحدث عندما يضغط المستخدم الـ single ASCII أو

الـ control key ولكن ليس مفتاح له وظيفة أو أى مفتاح خاص . استخدم هذا الـ event فى حالة الضغط على حرف من لوحة المفاتيح وهذا الـ event يكون بعد الـ OnKeyDown ولكن قبل الـ OnKeyUp . قم دائماً بتحديد الـ KeyPreview بـ True عند استخدام هذا الـ event .

● **OnKeyUp** : يستخدم عندما يترك المستخدم أى مفتاح ، بما فى ذلك

المفاتيح التى لها وظيفة والمفاتيح الخاصة . الـ OnKeyUp event يتبع حدوث الـ

**OnKeyDown** استخدم هذا الـ event بمفرده مع الـ **OnKeyDown** لتنظيم نشاط لوحة المفاتيح - على سبيل المثال، فى لعبة تؤدي اعمال (من المحتمل من خلال الـ Timer) وبينما يستمر المستخدم فى ضغط مجموعة مفاتيح معينة. ويحدث الـ **OnKeyUp** لكل **OnKeyDown**، ولكن ليس من الضروري ان يكون هذا فى نظام صارم متعاقب تحت - فوق - تحت - فوق (يرسل الـ Windows دائماً رسائل مفاتيح تحت ومفتاح فوق بالترتيب ولكن ازواج الرسائل قد لا تكون متعاقبة - تحت ١، تحت ٢، فوق ١، فوق ٢ قد يحدث) قم دائماً بتحديد الـ **KeyPreview** بـ **True** عند استخدام هذا الـ event.

● **OnMouseDown**: يحدث عندما يضغط المستخدم أى زر الفأرة ويتم وضع مؤشر الفأرة فوق الـ client area للـ form ويستدعى الـ Delphi الـ **GetCapture** الـ Windows API للـ **OnMouseDown** لضمان ان الـ object يتلقى الـ event الـ **OnMouseDown** و **OnMouseUp** المتلاحقة. وهذه السمة يتم التحكم فيها بواسطة الـ **csCaptureMouse** فى خاصية الـ **ControlState** لـ **TControl** class.

● **OnMouseMove**: يستدعى عندما يحرك المستخدم مؤشر الفأرة داخل الـ client area للـ form. يمكنك ايضاً تحديد ما اذا كان المستخدم قد ضغط مفتاح الـ Shift، أو Alt، أو Ctrl اثناء تحريك الفأرة.

● **OnMouseUp**: يستدعى عندما يترك المستخدم زر الفأرة الذى قد انتج سابقاً الـ **OnMouseDown** event، بغض النظر عن وضع مؤشر الفأرة.

● **OnPaint**: يتم استدعاه عندما تحتاج محتويات الـ form الى عملية الـ updating - على سبيل المثال، بعد ان يقوم المستخدم بتحريك نافذة اخرى جانباً. لمزيد من المعلومات عن برمجة الـ Graphics، انظر الباب الثالث عشر، "تطوير تطبيقات الـ Graphics".

● **OnResize**: يستدعى بعد ان يتغير حجم النافذة وقد يكون هذا استجابة لسحب زر حجم النافذة، وتصغير أو تكبير النافذة. لاحظ ان النافذة يتم إعادة تحديد حجمها بالفعل عندما يتلقى تطبيقك هذا الـ event.

## الباب الثالث : معلومات أولية حول الـ Forms

● **OnShow**، يستدعى قبل ان تصبح النافذة مرئية . استخدم هذا الـ event. لتنفيذ ما تريده عند ظهور النافذة .

للتعامل مع مفتاح الـ Tab من خلال الـ OnKeyDown، راجع باستمرار الـ Shift flags. اذا ضغط المستخدم الـ Alt+Tab للانتقال من تطبيق الى آخر، يتلقى الـ OnKeyDown مجموعة المفاتيح هذه. اذا لم تتأكد من الـ Shift flags والذي يحتوى على Alt فإن برنامجك قد يتداخل مع نظام التشغيل الـ Windows.

## Form Templates

يمكنك إقامة الـ form الخاصة بك من لا شئ، أو يمكنك الاختيار من واحدة من عدة form templates، سابقة البرمجة يتم تقديمها مع Delphi. وهذا templates يسهل استخدامها وتعديلها، وهى متميزة جداً خاصة فى إنشاء انماط غوجية وفى التشغيل.

كما سيتضح فى الفقرات التالية، يمكنك ان تنشئ form templates خاصة بك لتوفر قاعدة مشتركة لبرامج جديدة مثل شاشات إدخال البيانات custom dialog boxes، وتطبيقات أخرى.

## استخدام الـ form templates :

قم بتجربة الخطوات التالية لإنشاء الـ About dialog باستخدام الـ form templates سابقة البرمجة تعرض معلومات حول حق الطبع ومعلومات أخرى عن تطبيقك.

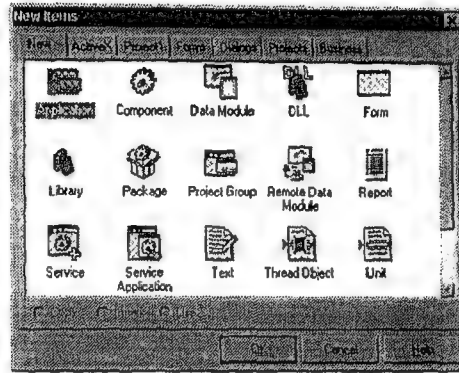
**ملاحظة:** ان النسخ الأولى من Delphi تتطلب اختيار الـ Options|Environment وتشغيل خيار الـ Gallery. استخدم الـ New Form، لإحضار الـ dialog من الـ form templates المتاحة والنسخ الاحدث من Delphi تعرض تلقائياً Dialog مشابهاً، رغم انه اطول، عندما تختار أمر الـ New.....

١- ابدأ مشروع جديد، وقم بإنشاء دليل مثل C:\Projects\Aboutex لللفاته. قم بتغيير الـ Name للـ MainForm form والـ Caption الخاص به ليكون About Box Example. احفظ المشروع فى الدليل الذى انشأته لجعل اسم الـ unit Main والمشروع بـ Aboutex.



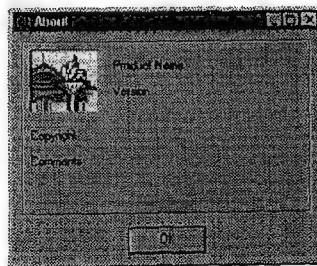
٢- لإضافة form ثانية للمشروع، اختر FileNew....As الموضح في شكل (٢-٣)، وهذا يعرض ال New Items dialog (والذى يحمل عنوان Browse Gallery) والذى يضع قائمة بال form و templates اخرى. يوضح جدول (١-٣) ال templates المختارة فى New Items dialog - سوف ترى البعض الآخر خلال هذا الكتاب.

٣- اضغط ال Forms page tab فى نافذة New Items dialog. اختر ال About Box template بالفأرة أو اضغط لل Tab أو مفاتيح الاسهم. اضغط ال Enter أو اضغط OK لإضافة ال form فى التطبيق واغلق ال New Items dialog.



شكل (٢-٣): New Items dialog form و templates أخرى

٤- يجب ان ترى نافذة About dialog الخالية الموضحة فى شكل (٣-٣). حسب النظام الافتراضى، تعتبر خاصية ال Name لل form محددة بـ AboutBox. قم بتغيير هذا ال Name ليكون AboutForm. وقم بتغيير ايضاً ال Caption الخاص بال AboutForm ليكون About This Program، أو الى أى نص آخر.



شكل (٣-٣): نافذة form-templates AboutBox الغير معدلة

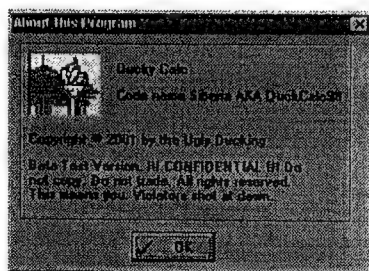
## الباب الثالث : معلومات أولية حول الـ Forms

**ملحوظة:** اننى اقترح إحتواء اسم الـ component ، كما يحدث فى الـ form بالنسبة للخاصية Name التابعة له .

Note

٥- احفظ المشروع ، وعندما يعرض الـ Delphi save dialog ، إضف الـ About لإسم ملف AboutForm unit . هذا ينشئ الـ Object Pascal module جديد ، وهو About.pas ، فى الدليل الحالى . ان للبرنامج الآن أثنان من الـ form واحدة للنافذة الرئيسية للبرنامج(MainForm) وواحدة للـ About dialog (AboutForm) . وكل form لها Object Pascal module تابعة لها : Main.pas للـ MainForm و About.pas للـ AboutForm . بالإضافة الى ذلك ، فإن الملف Main.dfm يخزن خصائص الـ MainForm; والـ About.dfm يخزن خصائص الـ AboutForm . قد تريد ان ترى إسماء ملفات المشروع مع Windows Explorer فى هذه المرحلة .

٦- قم بتعديل الـ Label component objects الخاصة بالـ AboutForm كما تريد . إدخل اسم برنامجك ، وملحوظة حق الطبع ، وأى نص آخر تريده . يمكنك ايضاً إضافة Label إضافي و components اخرى فى نافذة الـ AboutForm . وأية تغييرات تقوم بها تؤثر على هذا التطبيق فقط - فلا تتأثر الـ form template ، ويظل صالحاً للاستخدام فى المستقبل . يوضح الشكل (٣-٤) مثال من الـ AboutForm dialog التام .



شكل (٣-٤) ، الـ AboutForm التام من تطبيق الـ AboutEx

٧- اذا قمت بتشغيل البرنامج فى هذه المرحلة ، فإنه يعرض نافذة فارغة فقط . رغم ان الـ MainForm يصبح مرئياً بطريقة تلقائية ، فإن الـ AboutForm dialog يختفى لحين الحاجة إليه . لعرض dialog ، اضف أولاً للـ Button

component للـ MainForm . احتفظ بالـ Name الافتراضى ، وهو Button1 ، وقم بتغيير الـ Caption ليكون "Click Me!" ، أو أى نص آخر . حدد حجم الزر ونافذة الـ MainForm كما ترغب . اضغط مرتين الـ Button1 object فى الـ form لإنشاء الـ OnClick event handler ، واضف هذه العبارة بين البداية والنهاية :

AboutForm.ShowModal;

٨- وتقوم العبارة السابقة بإستدعاء ShowModal function التابعة للـ AboutForm . وهذا يعرض modal dialog ، وهو الذى يجب عليك إغلاقه قبل الاستمرار فى استخدام التطبيق ، و modeless dialog يسمح للقوائم والأوامر الأخرى بأن يعمل بشكل طبيعى بينما يظل الـ dialog مفتوحاً . يوضح الباب عشر الكثير عن الـ modal والـ modeless dialogs .

٩- إذا حاولت إجراء عملية الـ compile لتطبيق ، فسوف تتلقى رسالة الخطأ لان الـ Main unit لم تتعرف بعد على الـ AboutForm object فى About module . جرب هذا- ان النسخ الحديثة من Delphi يمكنها أن تضيف البرمجة اللازمة لاستخدام الـ AboutForm unit . أجب بنعم Yes اذا سؤلت . أو ، لإدخال البرمجة يدوياً ، اضغط الـ Cancel بنافذة الخطأ واطهر Main unit فى code editor (يمكنك الضغط على Ctrl+F12 واختر Main من View Unit) . حدد موضع أمر الاستخدامات فى أعلى الـ Main ، وهو يضع قائمة بالـ modules الأخرى التى تستخدمها هذه الـ unit وبين اسم الـ modules الأخيرة والفصلة المنقوطة ، ضع فصلة واكتب كلمة About . تأكد من أن الأمر مازال ينتهى بفصلة منقوطة على سبيل المثال ، إن الـ uses directive الخاص بك قد يبدو مثل ما يلى ، ولكن وعدد الـ modules بالطبع قد يختلف اعتماداً على التعبيرات التى قمت بها فى نافذة الـ AboutForm . والمدخل الجديد موضح هنا بخط سميك :

uses

Windows, Messages, SysUtils, Classes, Graphics,  
Controls, Forms, Dialogs, StdCtrls, About;

### الباب الثالث : معلومات أولية حول الـ Forms

١٠- والآن، بما أن الـ Main يدخل، أو يستخدم About unit، فإن العبارة السابقة التي تستدعي الـ ShowModal. اضغط F9 لإجراء عملية الـ compile، والـ link لتنفيذ التطبيق. عندما تظهر نافذته الرئيسية، اضغط الزر لعرض الـ AboutForm dialog. أغلق الـ dialog والبرنامج وارجع إلى Delphi.

يمكنك استخدام خطوات مشابهة لبرمجة أمر قائمة يعرض الـ AboutForm. بدلاً من الـ Button أضف الـ MainMenu component على الـ MainForm واضغط الـ object مرتين لإنشاء أوامر القائمة. اختر أمراً مثل Help>About في نافذة الـ form لإنشاء event handler، وأدخل نفس العبارة التي كتبته لـ OnClick الخاص لـ Button1.

وطريقة أخرى لإضافة unit داخل unit أخرى هي إضافة directive uses جديد في قطاع implementation للـ compiled unit. وهذا ما يفعله Delphi تلقائياً إذا أجبت بـ Yes في الـ error dialog في الخطوة رقم (٩). وللقيام بهذا يدوياً، احذف اسم الـ About unit، والفصلة التي تسبقها من الـ Main unit uses directive التي قمت بتعديلها في الخطوة رقم (٩). ضع كلمة implementation (أي تنفيذ) في نفس هذه الـ unit، وتحت هذا السطر مباشرة، أضف uses directive لإدخال الـ About module. يجب أن يبدو كلا الطرفين كما يلي:

```
implementation
uses About;
```

والاختلاف الرئيسي بين هذا الـ method وإضافة الـ Main uses directive module's في أعلى تلك الـ unit هو أن البنود الموجودة في القطاع uses للـ unit تكون خاصة بهذه modules إن استخدام units في تنفيذ الـ modules يساعد أيضاً في منع الأخطاء المسماة circular errors، والتي تسببها اثنان من الـ unit تستخدم كلاهما الأخرى. وهذا أمر مسموح به إلا أنه يتطلب أن تكون لكل منهما أو uses directive في قطاع implementation الخاصة بهما، Circular unit references من خلال قطاع الـ uses للـ unit غير مسموح به. ولا يهم عدد الـ units أخرى، فنسخة واحدة فقط من كل unit يتم تحميلها في الذاكرة. إن قطاع

ال uses تجعل تعريف متغيرات ال unit متاحة حتى يمكن للبرنامج أن يرجع للمتغيرات ويستدعى ال method ويستخدم بنود أدخلت :

جدول (١-٢) Item Templates الجديدة المختارة

الوصف	المنته	Template
يبدأ تكوين تطبيق جديد؛ كما هو في اختيار أمر ال FileNew Application	جديد	Application
يزودك بمعلومات حول component جديد تريد إنشاءه وحتى تعلم الكثير عن البرمجة في Delphi، فإنك لن تستخدم هذا ال template	جديد	Component
Form خالية ذات أغراض عامة؛ كما هو الحال في اختيار أمر ال FileNew Form. استخدم هذا template لإنشاء نافذة رئيسية للبرامج، child windows و custom dialog boxes	جديد	Form
تبدأ Unit جديد ليس له form تابعة بها. استخدم هذا ال template لإنشاء units لديها توضيحات، procedures، functions تريد أن تشارك فيها ال unit الأخرى	جديد	Unit
يعرض ملحوظة حق الطبع، ورقم النسخة، ومعلومات البرنامج الأخرى، دائماً ما يعرض استجابة لأمر ال Help>About	Forms	About box
قائمتان متداخلتان، ذاتا أزرار لتحويل البنود المختارة من إحدى القائمتين الى الأخرى. إنهما ذاتا فائدة في الاختيار من بين خيارات ملفات، ومواضع المعلومات الأخرى	Forms	Dual list box
dialog box مع أزرار OK، Cancel، و Help، وهي GroupBox area تستطيع حمل components أخرى، ومجموعة من ال page tab الظاهرة في الأعلى تغيير في حالة	Forms	Tabbed pages

استخدام complex dialogs التي بها controls كثيرة لتتناسب داخل شاشة واحدة، مثل الـ Options dialog. يستطيع المستخدمون اختيار page tabs لرؤية اجزاء اخرى من dialog.

Dialogs Standard Dialog يشبه الـ form الحالية، ولكن له ازرار OK و Cancel وهذه الفئة لـ templates قليلة متشابهة تضيف ازرار Help وتضع جميع الازرار في اسفل أو في الجانب الايمن من النافذة.

Dialog Dialogs Password صغير له مربع ادخال، وازرار OK و Cancel استخدمه لطلب كلمة السر من المستخدمين. كلمة المرور لا تكون مرئية اثناء الكتابة.

**فكرة:** فكرة في القطاع interface (الى أعلى الـ unit) على انه مساحة Tip  
عامة تكون مرئية للـ unit's الأخرى. فكر في قطاع implementation (الى الاسفل) على انه مساحة خاصة يحدث فيها برمجة العمل. الخاص بالـ unit. ان الـ unit's تستطيع استخدام هذه العناصر فقط types، والمتغيرات، والـ method، والـ declarations والتي تظهر في قطاع الـ interface التي تم ادخالها. وتستطيع الـ unit استخدام العناصر التي تم تعريفها في قطاع implementation وبصورة عامة، باستثناء تعديل الـ unit uses directive، يجب ان تكتب برمجة جديدة في قطاع الـ unit's. يقوم Delphi تلقائياً ببرمجة قطاع الـ interface للـ unit وأية تغييرات تقوم بها في هذا القطاع قد تتعارض مع قابلية الـ Delphi لإنتاج الـ code.

**فكرة:** عندما تقوم لأول مرة بحفظ مشروع ذي forms متعددة، يقدم Tip  
Delphi مجموعة من الـ save-as dialogs، واحداً لكل modules لم يتم تسميتها بعد. في احسن الحالات، قد يكون هذا محيراً في أسوأ الحالات، قد يؤدي بأكثر المبرمجين خبرة ومهارة الى الجنون. وللحماية من التسمية الخطأ للملفات الـ modules والذي قد يحدث دون قصد في التطبيقات التي لها form متعددة، قم بإنشاء كل forms جديدة منفردة. إضف الـ Name والـ

Caption التابعين لها، واختر File|Save All لتسمية وحفظ modules الجديدة. سوف يستغرق هذا وقتاً أطول لحفظ كل forms جديدة، ولكن هذا الأسلوب يعتبر أقل مخاطرة من حفظ forms متعددة في وقت واحد.

### إنشاء Object Repository templates

إن إنشاء ال form و application templates الخاصة بك تعد طريقة عظيمة للمشاركة في البرمجة أو لتطوير software shells. على سبيل المثال، إذا كان مستخدمى برنامجك يطلبون شاشات إدخال بيانات متعددة تختلف في نواحي قليلة، فيمكنك اختصار الوقت بإنشاء templates للعناصر المشتركة العامة. إن فرق وضع البرامج يمكنها أيضاً اختصار الوقت والمجهود لتطوير templates للمشروعات الجديدة، و demos و prototypes.

يمكنك حفظ أى form، أو حتى تطبيق بأكمله، كأنه template. ويقوم بهذا عن طريق إضافة form أو تطبيقك في ال Object Repository الخاص بـ Delphi. وهى طريق سهلة وبسيطة، وهى عظيمة في تخصيص Delphi على تصميمات template. إتبع هذه الخطوات لإضافة أى form أو تطبيق لل Object Repository :

**ملحوظة:** إن القرص المدمج الخاص بالكتاب لا يحتوى ملفات لما يأتى. استخدم هذه الخطوات لتحويل ال form، والتطبيقات، الى template.

Note

١- قم بتطوير وحفظ تطبيقك، أو افتح أى مشروع موجود. إحضر نافذة ال form المنشورة (اضغط Shift+F12، إذا لزم الأمر، أو اختر امر ال View|Form). يجب أن تشاهد نافذة ال form وليس ال code editor، لإضافة ال form لل Object Repository.

٢- اضغط Alt+F10 أو اضغط مرة واحدة زر الفأرة الايمن بينما يكون المؤشر داخل نافذة ال form المنشورة. هذا يؤدي لعرض قائمة اختر امر ال Add To Repository.

٣- لقد توفر لديك Title، Description، Page (اختر واحدة من القائمة، أو إضف اسم page، جديدة)، ومعلومات Author اختيارية. اضغط زر Browse لاختيار ايقونة لعرض template خاص بك، إما واحدة من دليل

### الباب الثالث : معلومات أولية حول ال Forms

Images\Icons الخاص و Delphi، أو واحد، من تصميمك . عندما ترضى عن إدخالك، اضغط OK . ان ال template الآن فى ال Object Repository ، ويظهر فى ال New Items dialog عندما تختار امر ال File\New الخاص Delphi .

إذا كنت تريد إنشاء template لتطبيق بأكمله - وهى طريقة انيقة لتوفير عناصر تطبيقات كنقطة بداية للبرامج الجديدة - افتح ملف مشروع التطبيق واختر امر ال Project\Add To Repository . أدخل معلومات واختر ايقونة كما فعلت فى الخطوة رقم (٣) . يمكنك الآن اختيار هذا ال template باستخدام File\New.... لتبدأ تطبيق جديد باستخدام العناصر السابقة .

**ملحوظة:** يقوم ال Object Repository بتخزين معلومات حول template . وتعتبر البيانات الحقيقية وملفات ال source code هى الاصول . ان إضافة ال form أو تطبيق لل Object Repository لا ينسخ ال source code و ال form فى Delphi ، إنه ينشئ مدخل يشير الى الملفات الاصلية . وكنتيجه لذلك ، بعد إضافة form أو تطبيق لل Object Repository ، فإن اية تغييرات تقوم بها فى الملفات الاصلية تؤثر على template فى الاستخدام فى المستقبل عن طريق ال File\New.... والميزة من هذا هو انه يمكنك تعديل ، إزالة اخطاء ، وتحديث template دون الحاجة لازالة أو إعادة تحميلها فى ال Object Repository .

ان ال source files لل templates الافتراضية ل Delphi تكون فى دليل ال Objrepos فى ال Delphi installation الخاص بك . يمكنك عرض هذه الملفات لتعديل ال templates ولكن قد تريد الاحتفاظ بنسخ من الاصول . يمكنك ايضاً عرض ال template entries فى New Items dialog باستخدام امر ال Tools\Repository..... . استخدم ال display الناتج لعرض ال objects فى كل صفحات ال New Items ، لحذف ال templates (لم تغيير source files) ، ولإعادة تسمية page tab .

وفى بيئة مشتركة ، مثل الشبكة التى لها واضعى برامج كثيرين يحتاجون الوصول الى ال application template و ال form ، فإنك تحتاج ان تقيم مسارا



يدعى "Shared Repository Directory". يتم تخزين هذا المسار فى سجل ال Windows. لتغييره، استخدم امر ال Tools|Environment Options، واختبر Preferences page tab. ادخل اسم مسار مشترك حيث يبحث Delphi عن ملف ال Object Repository، وهو DELPHI32.dro. وهذا المسار غير ذلك المذكور فى الفقرة السابقة. ان المسار الافتراضى هو دليل تركيب \bin الخاص بـ Delphi وانسخ ملف DELPHI32.dro الافتراضى فى دليلك المشترك اذا قمت بتغيير اسم مساره.

**ملحوظة:** ان ال Delphi control متصل الى ملف ال Object Repository المشترك باستخدام ملف إغلاق يسمى .DELPHI32.drl. وهذا قد تم صنعه لمنع اثنين أو أكثر من المبرمجين من الكتابة فى ال Object Repository فى وقت واحد. رغم ذلك، اذا خرج Delphi بشكل غير طبيعى اثناء تعديل ال Object Repository، قد لا يتم إزالة إغلاق الملف. لاستعادة إمكانية التوصل الكامل للـ Repository، ببساطة احذف .DELPHI32.drl.

### حذف Object Repository templates

اتبع الخطوات التالية لحذف ال application template أو form من ال Object Repository:

١- اختر .....Tools|Repository لاحظ نافذة ال Object Repository.

٢- اختر اسم ال page التى يتم تخزين ال template به - إنه نفس الاسم page tab المعروض عن طريق امر ال File|New.... ان ال template لهذه ال page معروضة فى نافذة ال Objects.

٣- اختر ال templates الذى تريد إزالته واضغط زر ال Delete Object. اجب بـ Yes اذا كنت تريد إنهاء الحذف.

٤- يمكنك أيضاً حذف ال page كاملة من ال templates بواسطة زر ال Delete Page.

ان حذف الـ template لا يحو الملفات الخاصة به . إنهم يقولون في ادلتهم الاصلية . لاستعادة الـ template الذى تم حذفه ، قم باضافته مرة اخرى للـ Object Repository . قد تحذف الـ form template الأساسية خاصة بالـ Delphi ولكن هذه تعتبر Objects مفيدة للغاية ولا يجب ان تحذفها دون سبب وجيه . ان بعض الأمثلة فى هذا الكتاب تشير الى ان واحدة أو اكثر من الـ template القياسية متاحة . اذا لم تستطيع إيجاد template مذكوراً ، فمن الممكن ان يكون احدهم قد قام بتعديل الـ Object Repository الخاص بك ، وعليك إما ان تعيد انشاء أو تعيد إنزال Delphi .

### استخدام الـ Dialog للنافذة الرئيسية:

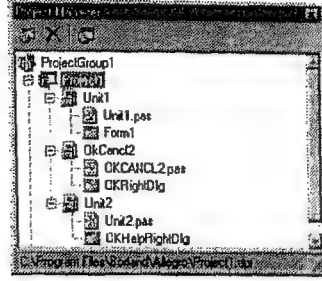
فى بعض الحالات ، قد تريد تغيير النافذة الرئيسية لـ forms مختلفة . على سبيل المثال ، من المفيد عادة استخدام dialog أو form template اخرى كنافذة تطبيق . وواجهة التطبيق الناتجة تكون نظيفة وبسيطة ، ولا يتطلب ذلك menu bar . ويقوم المستخدمون بتشغيل البرنامج بضغطة زر واختيار controls اخرى فى النافذة .

اتبع هذه الخطوات لتغيير النافذة الرئيسية لتطبيق لتكون form أو template اخرى :

١- ابدأ بتطبيق جديد .

٢- اختر الـ FileNew..... ، واختر الـ form template من نافذة الـ New Items dialog ، والمعروف أيضاً بالـ Object Repository . اختر Dialog page tab واختر الـ Dialog مع الـ Help form template .

٣- اختر الـ View|Project Manager [انظر شكل (٣-٥)] . أبرز الـ Project1 entry ، واضغط الزر الايمن للفأرة لظهور القائمة اختر امر الـ Options وفى الـ Project Options dialog . الناتج ، اختر الـ Forms page tab ، وقم بتغيير المدخل فى الـ "Main form" edit box ليكون OKHelpRightDlg (يمكنك ان تقوم بذلك ببساطة باختيار اسم الـ object من القائمة) . اضغط OK لقد قمت الآن بتحديد الـ OKHelpRightDlg يجب استخدامه كـ form الرئيسية للتطبيق .

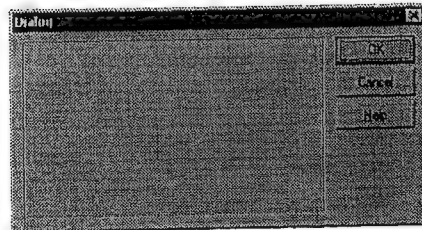


شكل (٣-٥)، نافذة ال Project Manager الخاص بـ Delphi

٤- يجب ان تعرض نافذة ال Project Manager مرة أخرى . استخدم هذه النافذة لحذف ال form الرئيسية القديمة للتطبيق ، والتي لا حاجة لها الآن . لفعل هذا ، قم بابرار ال Unit1 في نافذة ال Project Manager ، اضغط زر الفأرة الايمن لتظهر قائمة ، واختر Remove من امر المشروع . اجب بـ No اذا حفرك Delphi على الاحتفاظ بـ Unit1.pas . هذا يؤدي الى حذف ال form الافتراضية للمشروع وملف ال source code unit الخاصة بها . وتختفى ال form القديمة ايضاً من العرض ، تاركة نافذة ال OKHelpRightDlg وملف ال source code فقط .

٥- اغلق نافذة ال Project Manager أو إبعدها جانباً . اضغط المشروع باستخدام File/Save All..... ، كما هو معتاد ، قم بتسمية ال main unit والمشروع بـ Test (الاسماء بالضبط لا تهم) .

٦- قم بعملية ال Compile ، وال link وتشغيل التطبيق بضغظ F9 . يوضح شكل (٣-٦) النافذة الرئيسية للبرنامج - dialog box - اذا ازرار Help و OK و Cancel . لانك لم تحدد اية event handler لأزرار ال dialog ، فإن الازرار لا تفعل شيئاً .



شكل (٢-٦): ال dialog template يستخدم كنافذة رئيسية

## إغلاق نافذة:

ان إغلاق نافذة بشكل سليم يعد أمراً هاماً تماماً كإظهار واحدة وتشغيلها .  
كما عرفت ، فى الـ source code ، يمكنك إغلاق أى Delphi form object باستدعاء الـ Close procedure الخاص بها . اذا كانت النافذة هى نافذة التطبيق الرئيسية ، فإن هذا ينهى البرنامج ايضاً . قم بإنشاء برنامج event handler بضغط الـ Button مرتين واستدع الـ Close كما يلى :

begin

Close; { <-- insert this statement here }

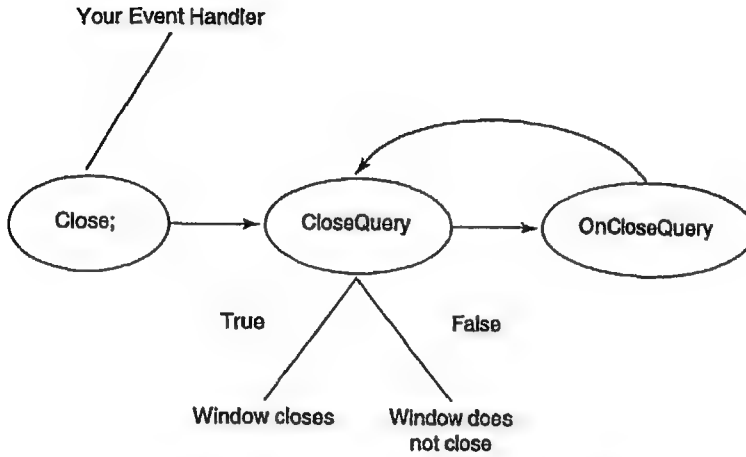
end;

رغم ان هذا يبدو بسيطاً ، فإن استدعاء Close يؤدي الى تشغيل اعمال كثيرة خلف الستار . لا يجب عليك برمجة هذه الانشطة -إنها تحدث بشكل تلقائى ، ولكن من المهم ان تفهم العملية داخلياً ، يقوم الـ Close باستدعاء function أخرى ، وهى CloseQuery ، والتي تعود True اذا كان امر إغلاق النافذة مقبولاً . اذا عادت الـ CloseQuery بـ False ، فإن استدعاء الـ Close يكون قد تم إلغاءه والنافذة لم تغلق . اذا عادت الـ CloseQuery بـ True ، يقوم البرنامج بتنفيذ OnClose (اذا كان هناك واحداً) لهذا الـ form .

ان تسلسل الاغلاق الخاص بـ Delphi يوفر فرص جيدة عديدة لأداء اعمال عندما تغلق الـ form داخلياً ، يقوم الـ CloseQuery بتنفيذ الـ OnCloseQuery ، والذي يمكنك استخدامه لتحديد ما اذا كان اغلاق نافذة الـ form أمراً آمناً . على سبيل المثال ، يمكن للـ OnCloseQuery الخاص لك فحص اذا ما كان المستخدم قد قام بحفظ البيانات الهامة . اذا لم يكن ، يمكن للبرنامج ان يعرض تحذيراً ، أو يمكنه إلغاء طلب إغلاق النافذة . وشكل (٣-٧) يوضح الاعمال التى تحدث عندما تستدعى الـ Close .

ويوضح مثلاً كيفية استخدام الـ OnCloseQuery لتأكيد نية المستخدم لإنهاء البرنامج باغلاق نافذته الرئيسية . ابدأ تطبيق جديد ، ثم اتبع الخطوات التالية :

- ١- اضعف Button component على الform . اضعف مرتين على ال Button واضف Close ؛ بين كلمتي البداية والنهاية الرئيسيتين لل event handler الناتج .
- ٢- اضعف F9 لإجراء عمليتي ال compile و ال link وتشغيل البرنامج . اضعف Button object للانهاء والعودة ل Delphi . هذا يعرض إغلاق الform عادي . بالتعمق داخل هذه العملية ، يمكنك إضافة جملة تحذير يسمح للمستخدمين بتغيير آراءهم بشأن إنهاء البرنامج .



شكل (٣-٧)؛ Delphi يؤدي هذه الاعمال عندما يستدعي البرنامج ال Close event للنافذة

- ٣- ومرة اخرى في Delphi ، اختر الform بالضغط في خلفيتها . تأكد ان ال Button1 objects لم يتم اختياره . اذهب الى ال Events page في نافذة ال Object Inspector ، واضغط مرتين مؤشر الفأرة في الحقل ، القيمة الى اليمين من ال OnCloseQuery . ادخل البرمجة الموجودة في القائمة (٣-١) . وكمراجع ، توضح القائمة ال event handler كاملاً - فقط ادخل العبارات الواقعة بين كلمتي البداية والنهاية الرئيسيتين . على القرص ، تجد هذه القائمة في دليل ال OnClose . وتعرض البرمجة message dialog له ازرار Yes أو No . اذا اختار المستخدم Yes ، فإن ال MessageDlg يرجع القيمة mrYes ، وتحدد عبارة if ال CanClose ب True . أما اذا اختار المستخدم No ، يحدد البرنامج ال CanClose ب False .

### الباب الثالث : معلومات أولية حول الـ Forms

٤- اضغط F9 لتشغيل البرنامج. هذه المرة، عندما تضغط زر النافذة، يطلب message dialog تأكيداً كما هو موضح في شكل (٣-٨). اختر Yes للإبقاء No لإلغاء طلب الاغلاق.

**ملحوظة:** إدخال بحذر البرمجة الموجودة في قائمة (٣-١) بالضبط كما هو موضح، أو إنسخها من ملف الـ CloseQuery.pas على القرص المدمج في دليل الـ Source\Misc. إذا ادخلت هذه، لاحظ الاقواس المربعة والاقواس الأخرى في هذا الـ code، والتي قد تبدو متشابهة في الصفحة المطبوعة.



شكل (٣-٨): message dialog للتحذير

القائمة (٣-١): OnCloseQuery event handler

```
procedure TForm1.FormCloseQuery(Sender: TObject;  
  var CanClose: Boolean);  
begin  
  if MessageDlg('End the program?',  
    mtConfirmation, [mbYes, mbNo], 0) = mrYes  
    then CanClose := True  
    else CanClose := False;  
end;
```

### Form Frameworks

تصف الاقسام التالية بعض الأشياء المنظمة التي تستطيع فعلها بالـ form، مثل إظهار scroll bars، جعل نافذة تبقى على قمة نوافذ أخرى وإنشاء الـ splash screen عند البدء.

## :Quick-and-dirty scroll bars

ان ال scroll يعد واحداً من اقدم الوسائل المعروفة لاستعراض المعلومات فى مساحة صغيرة نسبياً. واليوم، اصبح ال scroll واحداً من حيل العرض المقبولة فى نظم الحاسوب الحديثة لرؤية وثائق ضخمة على شاشات صغيرة. بالإضافة الى كونه اداة واجهة تطبيق مفيدة، يعد ال scroll bar رابطة بالماضى.

كما تعرف، ان scroll bar هو احد Windows controls التى تضغطه، تسحبه، تعرض المعلومات فى نافذة. اضغط زر ال scroll bar، لتعرض بسرعة بطيئة (سطر واحد فى المرة، مثلاً)، واضغط داخل ال shaded bar لتمر على البيانات اسرع. اسحب ال thumb box لتحصل على منظر آخر. على سبيل المثال، لتنتقل الى نقطة المتصف بالوثيقة، اسحب thumb box الى منتصف ال Bar.

فى ال Window 95، تطورت ال thumb box لتحدد كم المعلومات الغير معروضة فى النافذة - و ال thumb box كبير تحدد ان معلومات قليلة جداً فقط هى التى لا ترى، بينما يحدد thumb box صغير الى انه هناك معلومات كبيرة خارج نطاق الشاشة. لا يجب عليك اداء أية برمجة خاصة لتشغيل هذه الخاصية.

## إضافة scroll bar لـ form:

ان إضافة scroll bar لـ form يعد أمراً سهلاً، ولكن يمكنك عمل تنقيتات عديدة لل scroll بشكل اكثر سلاسة. اتبع الخطوات التالية لإضافة scroll bar لتطبيق ال BitView من الباب الثانى.

١- انسخ التطبيق BitView من الباب الثانى فى دليل جديد يسمى Mybits (أو أى اسم آخر). انسخ فقط ملفات Bitview.dpr، و Main.dfm، و Main.pas. لا تنسخ اية ملفات اخرى، خاصة مثل الملفات المكتبية التى تنتهى بـ .dsk. والتى قد تحتوى على اسماء مسارات لدليل ال Bitview الافتراضى. وكمراجع، فإن الملفات التامة لهذا البرنامج توجد فى دليل ال Bitview 2 على القرص المدمج.

٢- افتح مشروع ال Bitview (تأكد من التغيير الى الدليل الذى انشأته فى الخطوة رقم (١)). اضغط مرة واحدة ال outlined BitImage، والذي يملأ

### الباب الثالث : معلومات أولية حول الـ Forms

client area للـ form (أو، اختر BitImage من القائمة بالـ Object Inspector). وفي صفحة الـ Properties في الـ Object Inspector، اضغط مرتين خاصية الـ Stretch التابعة للـ bitmap لتغيير الـ True الى False. ومع كون الـ Stretch غير عاملاً، لم يعد البرنامج يقوم بتعديل bitmap لتناسب مع حجم الـ BitImage ولذا، فأنت تحتاج الى إضافة scroll bar لترى الصور أكبر من النافذة.

٣- حد خاصية الـ WindowState التابعة للـ MainForm بـ wsNormal، حتى يتم عرض النافذة في حجمها المصمم بدلاً من الشاشة كاملة. (اختر MainForm من القائمة الخاصة بالـ Object Inspector). وقم أيضاً بتغيير الـ Caption الخاص بالـ MainForm ليكون Bitmap Viewer مع Scroll Bars.

٤- لإضافة scroll bar أفقى للـ form، اضغط مرتين خاصية الـ HorzScrollBar التابعة للـ form لفتح قيمها الفرعية الخمس. اضغط اسم الخاصية، وليس قيمتها. حدد الـ Range بـ ١٠٠٠ وتأكد من أن الـ Visible في الـ True (وهي القيمة الافتراضية). عندما تضعها Enter أو تنتقل الى الـ Property أخرى، يظهر scroll bar أفقى في نافذة الـ form. اذا لم يظهر، قم بتحديد مدى أكبر.

٥- لإضافة scroll bar رأسى للـ form، أعد الخطوة السابقة مع خاصية الـ VertScrollBar للـ MainForm. وتظهر النافذة النهائية للـ form الآن scroll bars أفقى ورأسى.

٦- اضغط F9 لتشغيل التطبيق الذى تم تعديله. افتح أى ملف bitmap مثل الـ Sample.bmp فى دليل الـ Data. أعد تحديد حجم النافذة، واستخدم الـ scroll bar الخاصة بها لتحريك الصورة الى أعلى وأسفل ويميناً ويساراً.

بشكل عام يمكنك استخدام خطوات أبسط لإضافة scroll bars لأي component يقع خلال الـ client area للنافذة. وفي كل خاصية scroll bar، تأكد من:

- أن القيمة الفرعية Visible تكون True.



• أن القيمة الفرعية Range أكبر من خاصية ال ClientWidth التابعة لـ from . إن القيمة فيما بين ١٠٠٠ و ٢٠٠٠ تكون مناسبة غالباً .

إن scroll bar تفيد فى الأنماط النموذجية وفى إختيار عمليات التحريك؛ رغم ذلك، للحصول على تحريك أكثر هدوءاً، فسوف تحتاج أن تقوم بعدة إصلاحات للتطبيقات التامة كما هو موضح فيما يلى :

### ضبط ال scroll bar:

يتوقع مستخدمى ال Windows ذوى الخبرة أن تقوم ال scroll bar بالأداء . وفيما يلى بعض الخطوات التى اذا لم يتم إتباعها بدقة، فسوف تطلق السخرية من مستخدمى برنامجك :

- يجب ألا تظهر scroll bar حتى يفتح المستخدم وثيقة فى النافذة .
- إن ظهور scroll bar يجب أن يشير الى أن هناك مزيداً من المعلومات متاحة بالتحريك . على سبيل المثال، فى نافذة نص، يشير ال scroll bar الرأسى الى أن الوثيقة تحتوى سطوراً أكثر مما يمكن للنافذة أن تظهره .
- إن إعادة تحديد حجم النافذة، أو تحميل وثيقة أخرى، يجب أن يعرض أو يحذف ال scroll bar حسب الحاجة . على سبيل المثال، اذا كان تكبير نافذة يؤدي الى ظهور الوثيقة بأكملها، فيجب أن تختفى ال scroll bar . واذا قمت بتحميل وثيقة كبيرة، فإن ال scroll bar، يجب أن تظهر مرة أخرى اذا لزم الأمر .
- إن وضع الإشارة الخاص بـ scroll bar يجب ان يشير الى موضع الرؤية النسبى فى الوثيقة- أو، للتحديث بالمصطلحات الفنية، يجب ان يعكس مدى ال scroll bar حجم الوثيقة .

فى البرنامج التعليمى التالى، فسوف تقوم بكل التحسينات السابقة لتطبيق ال BitView . توضح القائمة (٣-٢) ملف Main.Pas form unit النهائى من دليل ال Source\Bitview2 على القرص المدمج .

افتح ملف مشروع ال BitView الذى تم تعديله من الفصل السابق، أو ابدأ بنسخ جديد للملفات المشروع dpr، dfm، و pas . ثم اتبع الخطوات التالية :

### الباب الثالث : معلومات اولية حول الـ Forms

١ - اضغط مرتين خصائص الـ VertScrollBar والـ HorzScrollBar التابعة للـ MainForm ، وحدد القيم الفرعية Range لكل خاصية بصفر مرة أخرى . وهذا يحذف الـ scroll bar من الـ form وتأكد من انها لن تظهر حتى يفتح المستخدم ملف الـ bitmap .

٢ - اختر page tab Events التابعة للـ MainForm في الـ Object Inspector . اضغط مرتين حقل القيمة المجاور الـ OnResize لإنشاء الـ event-handler procedure يسمى FormResize . يقوم البرنامج باستدعاء الـ procedure في أى وقت يتغير فيه حجم النافذة ، بغض النظر عن كيفية حدوث هذا العمل . أدخل العبارتين التاليتين بين البداية والنهاية (واضاً ارجع للقائمة (٢-٣) في نهاية هذا الفصل اذا أردت مزيداً من المساعدة) :

```
HorzScrollBar.Range := BitImage.Picture.Width;
```

```
VertScrollBar.Range := BitImage.Picture.Height;
```

٣ - تحدد العبارتان السابقتان خصائص الـ Range في الـ scroll bar الرأسية والأفقية الخاصة بالـ form لتكون مساوية لعرض وارتفاع الـ Picture الخاصة بالـ BitImage . ومع هذه التغييرات ، تعكس الـ scroll bar بصورة تلقائية حجم الصورة . كما ستكتشف ، ان هذه التغييرات وحدها لا تكفى لضمان التحريك الهادئ .

٤ - اضغط F9 لتشغيل التطبيق . افتح ملف الـ bitmap . في هذه المرحلة لا تظهر اية scroll bar حتى تنتهى من اعادة تحديد حجم النافذة . لعرض الـ scroll bar عند فتح ملف جديد فإن هذا يتطلب تعديل الـ FileOpen event handler . اترك البرنامج وارجع الى Delphi .

٥ - لكى تحدد الـ event handler الصحيح ، اختر FileOpen من نافذة الـ form ولان الـ procedure مازال موجوداً لهذا الأمر ، فإن Delphi يضع المؤشر على أول عبارة في الـ event handler . ادخل السطر التالى بعد العبارة التى تحدد اسم الملف خاصية الـ Caption بالـ form وهذا يكون داخل اعماق كلمتى البداية والنهاية الرئيسية . (راجع القائمة (٢-٣) في نهاية هذا الفصل اذا أردت مساعدة فى وضع العبارة) .

FormResize(Sender);

٦- اضغط F9 لتشغيل التطبيق . افتح ملف bitmap . تظهر scroll bar تلقائياً اذا لم تتناسب الصورة داخل نافذة البرنامج . قم بتغيير حجم النافذة اذا لزم الأمر وقم بتجربة عمل ال scroll bar .

ان العبارة التي اضفتها في الخطوة رقم (٥) تستدعي برنامج ال event handler موجود من داخل procedure آخر . في والامثلة الاولى عرفت كيفية تحديد أكثر من events لنفس ال handler ، مما يؤدي للتشارك في ال الموجود فيما بين عمليات متعددة . وهنا ، فإنك تستخدم اسلوباً آخر للتشارك في ال code ، ولكن في هذه الحالة ، فإنك قمت باستدعاء ال event handler من داخل آخر يؤدي اعمالاً إضافية- في هذه الحالة ، تحميل ملف bitmap من على القرص . ونتيجة للعبارة الجديدة ، فعندما تقوم بتحميل وثيقة ال bitmap ، يقوم البرنامج تلقائياً بتعديل مدى scroll bar ، والذي يؤدي الى ظهور أو اختفاء ال scroll bar حسب الحاجة .

**فكرة:** عند حساب حجم الصورة ، تأكد من استخدام خصائص ال Width والHeight الصحيحة . على سبيل المثال ، إن التعبير BitImage.Picture.Height يحصل على إرتفاع bitmap قد تم تحميله داخل BitImage component . والتعبير BitImage.Height يشير الى إرتفاع component object ، وليس الى إرتفاع البيانات داخل ال object ، وهي القيمة التي تريدها غالباً .

القائمة (٢-٢): Bitview2\Main.pas

unit Main;

interface

uses

### الباب الثالث: معلومات أولية حول الـ Forms

Windows, SysUtils, Messages, Classes, Graphics, Controls,  
Forms, Dialogs, Menus, ExtCtrls;

type

TMainForm = class(TForm)

  BitImage: TImage;

  MainMenu1: TMainMenu;

  File1: TMenuItem;

  Open1: TMenuItem;

  Exit1: TMenuItem;

  OpenDialog1: TOpenDialog;

  procedure Open1Click(Sender: TObject);

  procedure Exit1Click(Sender: TObject);

  procedure FormResize(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

MainForm: TMainForm;

implementation

```
{ $R *.DFM }
```

```
procedure TMainForm.Open1Click(Sender: TObject);
```

```
begin
```

```
  if OpenFileDialog1.Execute then
```

```
    begin
```

```
      BitImage.Picture.LoadFromFile
```

```
      (OpenDialog1.Filename);
```

```
      Caption := OpenFileDialog1.Filename;
```

```
      FormResize(Sender);
```

```
    end;
```

```
end;
```

```
procedure TMainForm.Exit1Click(Sender: TObject);
```

```
begin
```

```
  Close;
```

```
end;
```

```
procedure TMainForm.FormResize(Sender: TObject);
```

```
begin
```

```
  HorzScrollBar.Range := BitImage.Picture.Width;
```

```
VertScrollBar.Range := BitImage.Picture.Height;
```

```
end;
```

```
end.
```

### ListBox scroll bar السريعة:

تقدم هذا طريقة أبسط لإضافة scroll bars لنافذة. وهذا الأسلوب يفيد بشكل خاص في عرض القوائم الطويلة. قم بتجربة الخطوات التالية :

١- ابدأ مشروع جديد واضف ListBox component object من Standard palette الى الـ form

٢- اختر قيمة خاصية الـ ListBox object's Items واضغط الزر اليساوى لفتح Delphi's string editor. ادخل ٢٠ سطر نص أو ما الى ذلك (أو إنسخ بعض الاسطر من ملف نص). اغلق الـ editor. اذا اردت، استخدم خاصية الـ Font لاختيار حجم ونمط الـ font لنص الـ ListBox.

٣- حدد خاصية الـ Align للـ ListBox بـ alClient. هذا يدفع الـ ListBox object لئلا نافذة الـ form والتي قد تريد إعادة تحديد حجمها في هذا الوقت.

٤- اضغط F9 للتشغيل. يظهر تلقائياً scroll bar رأسى عندما تكون النافذة قصيرة جداً بحيث لا تستطيع عرض كل سطور النص فى ListBox control. (لقد أنشأت أيضاً الآن text viewer يسهل استخدامه).

ان الخطوات السابقة تضيف scroll bars رأسية فقط لـ ListBox component objects. لإنشاء scroll bars أفقى، ادخل العبارة التالية فى event handler على سبيل المثال، الـ OnActivate event للـ form.

```
ListBox1.Perform(LB_SETHORIZONTALEXTENT, 1000, 0);
```

تستدعى العبارة الـ VCL-internal Perform الموروثة لـ component مثل الـ ListBox من TControl class ويرسل الـ Perform رسالة، فى هذه الحالة LB\_SETHORIZONTALEXTENT، للـ control. والقيمة ١٠٠٠ هى عرض البيانات فى الـ ListBox مقدراً بالـ pixels وهو بالتقريب هنا.

## لقاء نافذة ال form الخاصة بك على القمة،

ان بعض البرامج - مثل الساعات ، system resource utilities ، Clacks toolbars - تعتبر اسهل في الاستخدام اذا بقيت فوق النوافذ الأخرى . وهنا نشرح كيف نحافظ على نافذة ال form تبقى دائماً فوق الأشياء الأخرى .

على القرص المدمج: ان ملفات البرنامج التعليمي التالي مخزنة على دليل ال Source\Ontop في القرص المدمج المرفق بهذا الكتاب .



١- ابدأ تطبيق جديد . قم بتسمية ال form بـ MainForm وحدد ال Caption الخاص بها بـ On Top Demo . احفظ المشروع في دليل مثل C:\Projects\Ontop . قم بتسمية MainForm's unit بـ Main و ملف المشروع بـ Ontop .

٢- اضع MainMenu object في ال MainForm . اضغط مرتين ايقونة ال object وقم بإنشاء قائمة تسمى Demo تكون ذات أمرين : Stay on top و Exit . إغلق ال Menu Designer .

٣- ارجع لنافذة ال form ، اختر امر ال Exit وادخل Close ، بين كلمتي البداية والنهاية الرئيسيتين . اختر امر ال Stay on top وادخل البرمجة الموضحة في القائمة (٣-٣) . تعد الخطوة الأخيرة من هذا البرنامج التعليمي سوف اشرح كيف تعمل هذه البرمجة . ان نص ال source code موضوع على القرص في ملف ال Main.pas في دليل ال OnTop .

٤- اضغط F9 لتشغيل البرنامج الكامل . اختر أمر ال Stay on top لتجعل النافذة تبقى فوق نوافذ التطبيقات الأخرى . لاحظ ان اختيار الأمر يضيف به علامة صح . اختر الأمر مرة أخرى لحذف هذه العلامة .

### القائمة (٣-٢): إظهار ال Stay on top

```
procedure TMainForm.Stayontop1Click(Sender: TObject);
begin
```

```
with Sender as TMenuItem do
begin
  Checked := not Checked;

  if Checked
  then FormStyle := fsStayOnTop
  else FormStyle := fsNormal;

end;

end;
```

ان ال procedure الموجودة في القائمة (٣-٣) تظهر إثنين من اساليب البرمجة الرئيسية ل Delphi . ان Sender parameter لل procedure يمثل شكل الامر الذى اختاره المستخدم . رغم ذلك ، فإن نوع procedure انما هو TObject ، والتي تركز عليها كل objects فى Delphi . إذن ، أى object يندرج من ال TObject يمكن تمريره لل procedure . لاستخدام هذا ال object فإن هذا يتطلب ان تخبر Delphi عن كينونته الفعلية .

وهذا قد تم هنا فى عبارة which ، والتي تقول ل Delphi ان يعامل Sender ، ك object TMenuItem . وهذا يمكن البرنامج من الوصول الى خصائص امر القائمة ، واحد مما هو ال Checked . ان تحديد هذا ال procedure بقيمته العكسية مع وجود التعبير not Checked يضع ويزيل علامة صح من القائمة . والاسلوب الثانى الذى تظهره القائمة (٣-٣) هى فى عبارة if التى تحدد ال FormStyle التابع لل from باحدى القيمتين - fsStayOnTop أو fsNormal . وتعتبر نفس خاصية ال FormStyle هذه متاحة فى نافذة ال Object Inspector ، ولكن لتجعل النافذة تبقى على القمة استجابة لامر المستخدم ، كما هو موضح هنا ، يجب ان يعين البرنامج بنفسه قيمة الخاصية وقت التشغيل .

وهذا يبرز نقطة هامة . بالإضافة الى تحديد قيم الخاصية باستخدام نافذة ال Object Inspector فى الوقت المحدد ، يمكنك ايضاً تعيين معظم قيم خصائص ال Object فى وقت التشغيل . على سبيل المثال ، قم بإنشاء



الـ procedure event handler، أو أضف Button object واضغط له مرتين .  
إضف الجملة بين كلمتى البداية والنهاية :

Caption := 'A New Window Title!';

عندما تقوم بتشغيل البرنامج، اضغط الزر لتغيير عنوان الـ form لاحظ ان  
العبرة يمكن ان تكتب ايضاً بهذا الشكل :

MainForm.Caption := 'A New Window Title!';

بالرغم من ان هذا يؤدى الغرض، الا انه يشير مباشرة لـ MainForm  
object، مما يجعل الـ code الناتجة اقل فائدة. من ناحية اخرى، إنها اكثر وضوحاً  
هكذا. ان العبرة الاولى يمكن ان تغير تعليق إية نافذة، أما الثانية فيمكن ان تغير  
تعليق الـ MainForm فقط. لتغيير نص الـ Button object، استخدم عبارة مثل :

Button1.Caption := 'Click Me Again!';

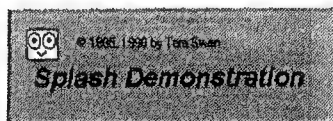
#### إنشاء الـ startup splash screen :

ان برامج الـ Windows المكتوبة بشكل جيد تعطى إشارة بدء، تسمى فى  
بعض الاحيان بـ splash screen. من خلال هذه الشاشة، يمكنك ان تضيف عامل  
هام لعرض برنامجك.

ولان المشروع نفسه ينشئ النافذة الرئيسية للبرنامج، ولان الشاشة يجب ان  
تظهر قبل ان يحدث هذا، فإن ملف المشروع يجب ان ينشئ البريق الخاص به. هذا  
يعنى انك يجب ان تقوم بتعديل الـ source code لملف المشروع، والتي تعتبر عادة  
غير مهمة فى برمجة Delphi.

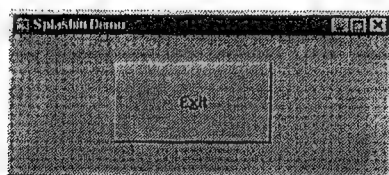
ان ملفات البرنامج التعليمى التالى موجودة فى الدليل الفرعى Splashin.  
ويوضح الشكل (٣-٩) شاشة البرنامج، والتي تظهر كنافذة ثابتة الحجم بلا حدود  
وبلا title bar، أو قائمة نظام، أو ازرار.

١- ابدأ مشروع جديد. اجعل اسم الـ form و MainForm وحدد الـ  
Caption الخاص بها بـ Splashin Demo. احفظ المشروع فى دليل مثل الـ  
C:\Projects\Splashin. اجعل اسم الـ MainForm's unit و  
Splashin اسم ملف المشروع.



شكل (٩-٣) شاشة البدء لل splash screen

٢- اصف ال Button component على ال MainForm. قم بتغيير خاصية ال Name لل Button لتكون ExitButton و ال Caption ليصبح Exit. (ملحوظة : اكتب E&xit فى ال Caption لتصميم الحرف x كمفتاح اختصار. يستطيع المستخدمون عندئذ ضغط Alt+X لاختيار الزر). ولمجرد الدعاية، قم بتكبير الزر، واعد تحديد حجم نافذة ال form كما هو موضح فى الشكل (٣-١٠) قم بإنشاء ال OnClick event handler الخاص بال ExitButton (يمكنك بسهولة ان تضغط مرتين Button object)، وادخل عبارة ال Close; بين كلمتى البداية والنهاية الرئيسيتين لل procedure



شكل (٢-١٠)، نافذة ال MainForm الخاصة بال Splashin's فى Delphi

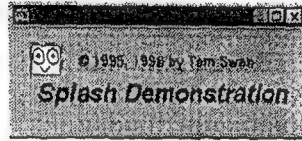
٣- اختر امر ال File|New Form أو اضغط زر New Form. وهذا يظهر ال Browse Gallery dialog، والذي يمثل عدة form سابقة البرمجة. (إذا لم تكن ترى ال dialog، استخدم ال Options|Environment استخدم Use on Blank form check box تحت عنوان Gallery). اختر Blank form template واضغط Enter أو اضغط OK لإغلاق ال Browse Gallery dialog.

٤- لقد أضفت لتوك form ثانية للتطبيق قم بتغيير خاصية ال Name لهذه ال form لتصبح SplashForm. احذف ال Caption الخاص بها. قم أيضاً بتغيير خاصية ال BorderStyle لها لتصبح bsNone وحدد الثلاث قيم الفرعية فى ال BorderIcons ب False. بعض هذه القيم اختياري وربما تود تجربتها مع أنماط، نوافذ مختلفة بعد ان تنهى هذا البرنامج التعليمي. يوضح الشكل (٣-١١) نافذة ال

SplashForm في Delphi التى سوف تنشئها فى الخطوات العديدة القادمة (ان المحتوى الدقيق للنافذة، رغم ذلك، شىء يرجع إليك).

٥- احفظ المشروع. عندما ينبهك Delphi الى اسم ملف ال unit فتأكد ان الدليل الصحيح هو الحالى (وهو C:\Projects\Splashin اذا كنت تتابع ما نشرح). ادخل Splash كاسم ملف ال SplashForm's unit.

٦- حدد خاصية ال SplashForm.L Enabled ب False. وهذه تعد إحدى الحالات النادرة التى لا تود فيها ان يتمكن المستخدمون من إعطاء أوامر الفأرة ولوحة المفاتيح لنافذة. فى هذه الحالة يريد ان يكون للبرنامج التحكم الكلى على عرض ال SplashForm.



شكل (٣-١١): نافذة SplashForm الخاصة ب SplashIn فى Delphi

٧- قم بإعادة تحديد حجم نافذة ال SplashForm. ولان النافذة ليس لها outline، اصف ال Bevel object من فئة ال Additional. هذا يساعد فى تعريف اطراف النافذة. حدد خاصية ال Align لل Bevel ب alClient. وقم ايضاً بتغيير Shape ليصبح bsFrame وال Style الخاص به ليكون bsRaised. وهذه القيم ترجع لك - جرب تحديدات مختلفة بعد ان تنهى البرنامج التعليمى.

٨- اصف component objects Label وال Image فى ال SplashForm. يمكنك ادخال اية عناصر رسومية تريد عرضها اثناء بدء البرنامج. رغم ذلك، لا تدخل اية ازرار أو controls متداخلة اخرى. ويظهر التطبيق نفسه ويزيل splash screen.

٩- هذه الخطوة اختيارية. حدد خاصية ال SplashForm's FormStyle ب fsStayOnTop، وقم بتغيير Position ليصبح poScreenCenter. ومرة اخرى هنا، تعتمد التحديدات على رغباتك، ولكن الفكرة الجيدة هى ان تضع splash screen فى المنتصف وان تجعل نافذتها دائماً فوق الأشياء والأخرى.

### الباب الثالث: معلومات أولية حول الـ Forms

١٠- اختر ViewProject Manager . قم بابرار مشروع الـ Splashin ، واضغط زر الفأرة الايمن لاطهار القائمة. اختر امر الـ Options . وفي الـ Project Options dialog الناتج، اختر Forms page tab . لاحظ ان MainForm و SplashForm موجودان في قائمة الـ Auto-create forms قم بابرار كل form واضغط زر السهم الايمن لتحريكها الـ Available forms . يتم إنشاء جميع Delphi forms في الذاكرة في وقت التشغيل مما يستهلك الذاكرة و system resources في الحالات المماثلة لهذه، حيث يقوم البرنامج بإنشاء form اثناء وقت التشغيل ، يجب ان تزيل هذه الـ form من قائمة الـ Auto-create . إغلاق نافذة الـ Project Manager .

١١- بعد ذلك ، قم بتعديل الـ code المشروع لعرض الـ splash form قبل ان تظهر النافذة الرئيسية . وهذه واحدة من الحالات النادرة التي تريد فيها ان تدخل عبارات في ملف المشروع . لعمل هذا ، اختر أمر الـ ViewProject Source . قم بتعديل العبارات بين البداية والنهاية ليلائم ملف مشروع الـ Splash.dpr في القائمة (٣-٤) . سوف اشرح المزيد عن هذه البرمجة بعد هذه الخطوات .

١٢- اذا قمت بتشغيل البرنامج في هذه المرحلة ، فإنه يعرض ويزيل startup splash بسرعة كبيرة بحيث لا تكون لديك فرصة ان تراه . لكي يظهر الـ dialog مرئياً لعدة ثوان ، اختر الـ MainForm للبرنامج (ذلك الذي له زر كبير) . قم بإنشاء لـ OnCreate select handler للـ form . قبل كلمة البداية الرئيسية ، اضع متغير longInt يسمى GetTickCount . بين البداية والنهاية ، اضع العبارتين لإستدعاء الـ Windows GetTickCount function ليحدد الـ Time الحالي لعدد الثواني الذي يقوم الـ Windows بالتشغيل وعبارة الـ while التي تتأخر لاربعة ثوان إضافية . (يرجع الـ GetTickCount عدد الملى ثانية الذي يقوم الـ Windows بتشغيلها . ان قسمة هذه القيمة على ١٠٠٠ يحولها الى ثوان) . إجمع للقائمة (٣-٥) ، Main.pas ، لترى هذه العبارات .

١٣- أضغط F9 للـ compile وللـ link وللتنفيذ ، سوف يعرض البرنامج الـ startup splash dialog لعدة ثوان قليلة ويختفي الـ dialog وتعرض الـ main window .

القائمة (٤-٣)، Splashin\Splashin.dpr

```

program Splashin;

uses
  Forms,
  Main in 'MAIN.PAS' {MainForm},
  Splash in 'SPLASH.PAS' {SplashForm};

{$R *.RES}

begin
  SplashForm := TSplashForm.Create(Application);
  SplashForm.Show;
  SplashForm.Update;
  Application.CreateForm(TMainForm, MainForm);
  SplashForm.Close;
  Application.Run;
end.

```

تعرض القائمة (٤-٣)، Splashin.dpr، كيف يُمكن لبرنامج إنشاء form object في وقت تشغيل. لفعل هذا، إستدع ال Create method في object's class، كما توضح هذه العبارة:

```
SplashForm := TSplashForm.Create(Application);
```

### الباب الثالث : معلومات أولية حول الـ Forms

تقوم هذه العبارة بإنشاء الـ object ، وتحديدده للـ SplashForm ، المعرف في الـ Splash.pas . يوجد الـ object الآن في الذاكرة ، ولكنه غير مرئى الآن ولكى يظهر وتحديث محتوياته ، إستدع Show method ، والـ Update للـ object :

```
SplashForm.Show;  
SplashForm.Update;
```

القائمة (٢-٥) : Splashin\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
Windows, SysUtils, Messages, Classes, Graphics, Controls,  
Forms, Dialogs, StdCtrls;
```

```
type
```

```
TMainForm = class(TForm)
```

```
ExitButton: TButton;
```

```
procedure ExitButtonClick(Sender: TObject);
```

```
procedure FormCreate(Sender: TObject);
```

```
private
```

```
{ Private declarations }
```

```
public
```

```
{ Public declarations }
```

```
end;
```

```
var
```

```
    MainForm: TMainForm;
```

```
implementation
```

```
{ $R *.DFM }
```

```
procedure TMainForm.ExitButtonClick(Sender: TObject);
```

```
begin
```

```
    Close;
```

```
end;
```

```
{ The following procedure pauses for a few seconds so  
  the Splash dialog remains visible for a predetermined  
  length of time. }
```

```
procedure TMainForm.FormCreate(Sender: TObject);
```

```
var
```

```
    CurrentTime: LongInt;
```

```
begin
```

```
    CurrentTime := GetTickCount div 1000;
```

```
while ( (GetTickCount div 1000) < (CurrentTime + 4) ) do
    Sleep(1);
end;

end.
```

يمكنك استخدام خطوات مشابهة لإنشاء وعرض أى form فى وقت التشغيل . رغم ذلك ، تأكد ان الـ form لم يتم إنشاءها تلقائياً . (استخدم امر الـ Options فى الـ Project Manager's للفحص) .

**ملحوظة:** ان استدعاء الـ Update بعد الـ Show ضرورياً هنا فقط لان البرمجة موجودة خارج الـ main message loop للتطبيق ، والذي يقوم بانتاج رسائل الـ Windows . ولا يتم إذن انتاج paint messages ، ولذا فإن استدعاء الـ Update يعد أمراً ضرورياً . وعادة ، يجب عليك ان تستدعى الـ Show لإظهار النافذة .

Note

بعد ان انتهيت من استخدام الـ form object ، قم بإخفائه من العرض ثم احذف الذاكرة التى يشغلها . وتصبح الذاكرة الحرة الآن متاحة لـ objects اخرى . على سبيل المثال ، بعد إنشاء النافذة الرئيسية (والتي تتوقف فى هذه الحالة لعدة ثوان حتى يمكنك رؤية الـ splash form) ، يستخدم برنامج الاظهار هذه العبيارة للتخلص من splashy dialog :

```
SplashForm.Close;
```

تظهر القائمة (٣-٥) ايضاً اساليب مفيدة للـ OnCreate event فى الـ MainForm . اولاً ، يقوم الـ procedure باستدعاء الـ GetTickCount ، وهى function فى الـ Window ترجع عدد الملى ثانية التى مرت منذ بدء الـ Windows . وهذه القيمة مقسمة على ١٠٠٠ تساوى عدد الثوانى المنقضية .

لتنوقف بضع ثوانى ، يخصص البرنامج القيمة المقسومة للـ CurrentTime ، وهو متغير من نوع الـ LongInt . ثم يستخدم الـ while loop تعبيراً يبدو معقداً لاختيار ما اذا كان الـ GetTickCount (مرة اخرى مقسوماً على ١٠٠٠) اقل من



الوقت الذى تم حفظه سابقاً زائداً أربعة. ويستدعى الـ while loop، الـ Windows Sleep function (القيمة المتغيرة (١) تحدد عدد الملى ثانية للـ sleep). وهذا يضمن ان العمليات الأخرى تتابع لها فرصة التشغيل بينما يحتل الـ CPU-intensive Loop العرض. بعد أربعة ثوانى، لا يستمر تعبير الـ while loop حقيقياً. وينتهى الـ procedure. وهذا يثبت ان الوقت يمر بالفعل بالرغم من جهودنا المضنية ان نبطئة- اذا فشلت هذه الحيلة، فلا تخف، ان العالم مازال بخير، ولكن ساعة حاسبك الآلى قد إنكسرت.

يتوقف برنامج الاظهار فقط لأنه لا يجد ما يفعله غير ذلك. اذا كان الـ OnCreate الخاص البرنامج يؤدي عدة بدايات، قد لا تحتاج الى إدخال وقفة صناعية. رغم ذلك، لقد اصبحت الحاسبات تزد أو سرعة كل يوم، والتعليمات التى تستغرق أربعة ثوان اليوم قد تنتهى فى المستقبل. للضمنان، استدع الـ GetTickCount لاختيار اذا ما كان على الأقل كمية معينة من الوقت قد مرت قبل إنها event handler. الخاص بالـ splash form.

## مقدمة حول الـ Data Modules:

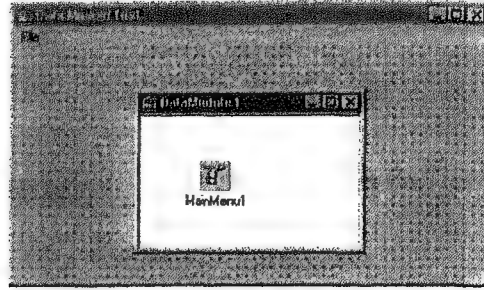
ان data module، تعتبر form منفصلة يمكن ان تحتوى nonvisual component objects. و Data modules لا ترى ابدأ فى التطبيق التام- فهى توفر نوعاً من المخازن للـ component التى قد تود ان تتشارك فيها مع units و forms اخرى للتطبيق وتعتبر الـ data module ايضاً مكاناً قريباً لحفظ nonvisual objects مثل الـ MainMenu والـ Timer. يمكنك استخدام data module لتحريك أيقونة الـ object خارج forms أخرى.

**ملحوظة:** تعتبر data modules مهمة بالأخص فى برمجة قاعدة البيانات كما هو موضح فى الباب السابع عشر، تطوير تطبيقات قاعدة البيانات. رغم ذلك، يمكنك استخدام data module فى أى تطبيق.

**على القرص المدمج:** اتبع الخطوات التالية لتعرف كيفية إنشاء واستخدام data module. يوضح الشكل (٣-١١) عرض الوقت المصمم للأمثلة فى Delphi. وتوجد الملفات التامة على القرص المدمج فى دليل الـ DMTest.

Note





شكل (١٢-٣): استخدام الـ data module لتحريك  
ايقونات مثل الـ MainMenu خارج visible form

١- إبدأ تطبيقاً جديداً. قم بتسمية الـ form الرئيسية بـ MainForm، وحدد الـ Caption الخاص بها بـ Data Module Test. احفظ كل الملفات، واجعل الـ Main.pas هو اسم الـ MainForm unit واسم مشروعها DMTest.dpr.

٢- اختر File | New Data Module. تظهر نافذة الـ form ذات خلفية بيضاء. يمكنك إدخال nonvisual component objects فقط في هذه النافذة. إذا حاولت ادخال visual component مثل الـ Button، فسوف تظهر لك رسالة خطأ.

٣- اختر File | Save All مرة أخرى. وعندما يظهر اسم الـ Unit2، أدخل الـ Module1 واضغط OK. وهذا يجعل اسم source file للـ data module's هو Module1.pas ويعرض أيضاً الـ label في نافذة الـ code editor. للتطبيق الآن أثنان من الـ forms.

٤- ادخل الـ MainMenu object من الـ Standard palette في نافذة الـ data module والتي تم تسميتها DataModule1. يمكنك تغيير هذا الاسم في تطبيقاتك، ولكن لهذا الغرض، فإننا نستخدم الاسم الافتراضي، حسب النظام.

٥- اضغط مرتين الـ MainMenu، قم بإنشاء قائمة File ذات امر Exit. اضغط مرتين امر الـ Exit لإدخال event handler له في الـ data module's source code أدخل العبارة التالية بين البداية والنهاية:

```
MainForm.Close;
```

٦- أجرى عملية Compile بضغط F9. سوف تتلقى رسالة الخطأ لان ال DataModule1 تشير الى ال MainForm، فى Unit اخرى. اما ان تدخل النص باستخدام ال Main فى قطاع data module's implementation، أو تجيب بـ Yes اذا سألت Delphi ان يقوم بهذا التغيير تلقائياً. (اذا كان باستطاعة Delphi تحديد أى من ال units ناقصة من قطاع ال uses، فباستطاعته إدخال اسمائها تلقائياً، ولكنه لن يفعل ذلك دون إذن منك).

٧- يمكنك الآن تشغيل التطبيق، ولكنه لا يعرض ال MainMenu فى نافذة البرنامج. لفعل هذا، فيجب ان تستخدم Main module وايضاً data module وقطاع ال uses آخر يمكنه ان يفعل هذه الخدعة. تأكد ان ال Main module يتم عرضها فى ال code editor، اختر File|Use Unit....، واختر Module1 من القائمة (هناك مدخل واحد فقط). اضغط OK لتضيف جملة Module1 uses فى قطاع Main implementation يمكنك كتابة هذه العبارة يدوياً اذا اردت.

٨- فى اخيراً، اضغط F12 لإظهار نافذة ال form الرئيسية واضغط بداخلها لعرض قيم خواصها فى نافذة ال Object Inspector. حدد خاصية ال Menu بـ DataModule1.MainMenu1 باختيار الاسم من قائمة اللائحة. وهذا يخبر ال form الرئيسية ان تستخدم ال nonvisual object، وهو MainMenu، الذى تم تخزينه فى DataModule1 object. اضغط F9 لتشغيل التطبيق التام.

قم بمراجعة الخطوات السابقة بحرص واختبر ملفات ال source code الموجودة فى القائمة (٣-٦) والقائمة (٣-٧). إنته جيداً للقطاع ال uses فى كل برنامج. ان هذا يعتبر مثلاً لـ circular unit reference والذى، كما ذكرت من قبل، يسمح له فقط قطاع ال uses من خلال قطاع ال implementation. لاحظ ان كل unit تشير للآخرى. من الناحية الفنية، ان استخدام ال data module يتطلب ان تستخدم ال unit المضافة فقط (وهى MainForm فى هذه الحالة) لكى تستخدم data module's unit. رغم ذلك، لكى يغلق ال MainMenu الخاص بـ MainForm module، فيجب ان يستخدم هو الآخر ال Main's unit. وهذا هو الأمر الطبيعى، و data module غالباً ما تستخدم هذا الاسلوب الدائرى.

## الباب الثالث : معلومات أولية حول الـ Forms

**Tip** فكرة: يمكنك ادخال الـ data module فى الـ Object Repository كما تفعل فى أى form اخرى . ولكن تأكد من انك اعطيت الـ module اسم منفرد (فلا تستخدم مثلاً اسمها الافتراضى DataModule). فهذا يمنع تداخل الأسماء فيما بين الـ form والـ modules المتعددة فى الـ Object Repository.

### القائمة (٢-٦) DMTestMain.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
    Windows, Messages, SysUtils, Classes, Graphics,  
    Controls,  
    Forms, Dialogs;
```

```
type
```

```
    TMainForm = class(TForm)
```

```
    private
```

```
    { Private declarations }
```

```
    public
```

```
    { Public declarations }
```

```
end;
```

```
var
```

MainForm: TMainForm;

implementation

uses Module1;

{ \$R \*.DFM }

end.

DMTest\Module1.pas : (۲-۷) القائمة

unit Module1;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs,

Menus;

type

TDataModule1 = class(TDataModule)

MainMenu1: TMainMenu;

procedure DataModuleFormExit1Click(Sender:

```
    TObject);  
    private  
    { Private declarations }  
    public  
    { Public declarations }  
    end;  
  
var  
    DataModule1: TDataModule1;  
  
implementation  
  
uses Main;  
  
{$R *.DFM}  
  
procedure TDataModule1.DataModuleFormExit1Click(  
    Sender: TObject);  
begin  
    MainForm.Close;  
end;  
  
end.
```

## الـ Splitter Component :

تحتوى الـ Additional palette على components متعددة تعتبر فريدة فى Delphi. وتعتبر هذه الـ controls مخصصة وهى إما ان تزيد من الـ controls المعيارية، أو تقدم خدمات من المؤكد انك سوف تحتاجها ولن تجدها فى تصميمات الـ controls الاصلية للـ Windows على سبيل المثال، ان BitBtn، الذى رأيته وهو يعمل وسوف تراه يستخدم مرة أخرى خلال هذا الكتاب، يزيد من استخدام زر الـ Windows فى الرسوم الجرافيكية للحصول على مظاهر أكثر متعة، ويعطى مزيداً من المعلومات.

وإحدى الـ component فى هذه الـ palette والذى اعتبره الأكثر إفادة هو الـ Splitter (class name TSplitter) استخدم هذا الـ component لإنشاء نافذة متعددة يمكن للمستخدمين تعديلها بالضغط والسحب. كثير من تطبيقات الـ Windows لها split windows مشابهة، وهذا الـ control يعد واحداً من الـ controls الأكثر إفادة. وهو أيضاً يسهل استخدامه. جرب هذه الخطوات :

١- إبدأ تطبيق جديد.

٢- قم بإسقاط الـ Memo من الـ Standard palette. على الـ form. حدد خاصية الـ Align لهذا الـ object بـ alLeft. هذا يؤدي الى محاذاة الـ Memo1 فى مقابلة الحافة اليسرى للنافذة.

٣- اضغط الـ Additional page tab وضع الـ Splitter component على الـ form، الى اليمين من Memo1. يظهر الـ control الى الحافة اليمنى من الـ Memo1 مثل المغناطيس والجديد. لاحظ ان خاصية الـ Align للـ Splitter1 هى الـ alLeft فهو نفس القيمة التى حددت للـ object.

٤- ارجع الى الـ Standard palette و قم بإسقاط Memo على الـ form الى اليمين من Memo1 و Splitter1 حدد خاصية الـ Align التابعة للـ Memo2 بـ alClient. فى اغلب الحالات، لابد ان تحدد قيمة الـ Align للـ panel الأخيرة الخاص للـ split window الى الـ client area لهذا الـ object سوف يملأ الحيز الباقى الى أعلى اليسار عن طريق تحديد أحجام الـ objects الأخرى.

### الباب الثالث : معلومات أولية حول ال Forms

قم بتشغيل التطبيق باستخدام F9 . اضغط وأسحب ال Splitter bar بين Memo components لإعادة تحديد حجمه .

قم بتجربة بعض الخيارات مع ال Splitter component . اختر Splitter1 من التطبيق الاختياري ، وقم بتجربة تحديدات هذه الخصائص :

● **Beveled** : حدد القيمة بـ True أو False للحصول على تعديل بسيط لمظهر ال splitter bar . وعن نفسى ، فإننى أفضل التحديد الافتراضى بـ True .

● **Cursor** : يجب أن تترك هذه عادة التحديد الافتراضى ، وهو crHSplit . لا يوجد سبب هام لتغيير مظهر المؤشر ، ولكن يمكنك استخدام مؤشر مختلف اذا أردت .

● **MinSize** : عين الحجم الأدنى لتحديد أطوال وعروض ال splitter bar . بالرغم من أن هذه القيمة محددة بـ ٣٠ حسب النظام الافتراضى ، فهى قيمة لا تعنى شيئاً ويجب تغييرها بموجب تحكم البرنامج - على سبيل المثال ، فى ال OnCreate الخاص بـ form . والصيغة الجيدة هى تعيين قيمة تساوى جزء من عرض النافذة - ١ / ٤ عرض ال client area ، مثلاً . تأكد من تحديث ال MinSize فى ال OnResize الخاص بـ Form ، بحيث اذا قام المستخدم بإعادة تحديد حجم النافذة بأكملها ، يتم تعديل الحجم الأصغر لل pane طبقاً لذلك .

● **ResizeStyle** : استخدم واحدة من الثلاث تحديدات الممكنة : rsLine لعرض خط مستقيم أثناء الضغط والسحب ليظهر أين سيكون انقسام النافذة عندما يترك المستخدم الفأرة ؛ rsNone لعدم عرض أية خطوط ؛ أو rsUpdate لإعادة تحديد حجم splitting objects باستمرار أثناء الضغط والسحب مثل ال bitmap التى تستغرق وقتاً فى إعادة رسمها . استخدم ال rsUpdate فى ال Memo وال objects الأخرى التى يمكن إعادة رسمها بسرعة . إننى لا استخدم أبداً ال rsNone لأنه يعطى إشارة بسيطة إلا لشكل المؤشر بأن عملية إعادة تحديد الحجم جارية .

● **Width** : إننى أفضل الواحد وهو أدنى قيمة هنا ، بالرغم من أن القيمة الافتراضية هى ثلاثة . ومع العرض واحد وال Beveled محدد بـ True ، يبدو splitter مثل حافة ال Memo أو أى object آخر أكثر من كونه control منفصل .



والمستخدمين الذين لديهم مشكلة فى استخدام الفأرة قد يجدون من الصعب التحكم فى splitter bar ، وفى تلك الحالة يجب عليك استخدام تحديد أكبر أو جعله اختيارياً وتعين خاصية ال Width فى وقت التشغيل بعبارة مثل :

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
    Splitter1.Width := StrToInt(Edit1.Text);
end;
```

يمكنك تقسيم النوافذ رأسياً وكذلك أفقياً- إن الأمر يكمن فى كيفية تحديد خصائص ال Align لل object . لتجربة هذا ، اتبع الخطوات السابقة ولكن قم بتغيير خصائص ال Align لل Memo1 وال Splitter1 لتكون alTop . يجب أن تكون خاصية ال Align لل Memo2 ، alClient ، قم بتشغيل التطبيق ، واضغط واسحب Splitter bar لإعادة تحديد حجم عرض ال Memo objects .

**Tip** فكرة: إن خاصية ال Height لل Splitter الأفقى تتحكم فى سمكه . وخاصية ال Width تتحكم فى سمك ال Splitter الرأسى :

### افكار للمستخدم المبتدئ

• فى بعض الأوقات ، قد يقرر Delphi أن ال event handler تم وضعه كمرجع ولكنه غير متوفر . ويطلب اذنا لحذف ال event . وعادة ، عندما يحدث هذا ، يجب أن نجيب بـ Yes ، وخاصة اذا كان ال event واحداً ممن حاولت التخلص منهم . ولكن اذا كان ال event واحداً ممن تعتقد أنه لا يجب حذفهم ، فقد تكون مسحت ال event handler بطريق الخطأ . أجب بـ No فى هذه الحالة ، واضغط ال event مرتين (أو اختر ال event handler من القائمة) لاستعادة ال procedure .

• استخدم File|Open لفتح ملف pas unit الخاص بال form لفحص برمجته . رغم ذلك ، فإن هذا لا يضبط ال form للمشروع الحالى . لفعل هذا ، اختر ال File|Add to project . ولا تظهر ال form بطريقة تلقائية . ولتظهرها ، اختر

### الباب الثالث : معلومات أولية حول الـ Forms

View/Forms أو استخدم الـ Project Manager ، والذي يمكن التوصل إليه من خلال قائمة الـ View . لإظهار الـ form unit ، اختر View/Units... .

● إذا كنت على دراية بـ text editors آخرين يعرضون الملفات في نوافذ منفصلة ، فبإمكانك اختيار نافذة Delphi editor قبل عرض modules أخرى . ولكن لا تفعل هذا . إن Delphi يستخدم نافذة editor واحدة لكل unit modules في المشروع ، وإغلاق هذه النافذة يغلق كل modules المفتوحة . رغم ذلك ، هذا لا يؤدي إلى حذف الملفات - إنه يغلقهم فقط في نافذة الـ editor . للانتقال إلى modules أخرى ، اختر View/Units أو اختر modules من page tab الخاص بنافذة الـ editor . بسبب أنه يحدد نافذة واحدة لكل modules ، يمكنك فتح كل unit في المشروع دون ازدحام العرض بعشرات النوافذ ، ويمكنك أيضاً تركها مفتوحة .

● إذا كنت تفضل أن يكون لديك نوافذ editor متعددة ، اختر View/New Edit Window .

● إن خصائص الـ Scroll bar لها قيم فرعية إضافية يمكنك استخدامها في الظروف الخاصة . لرؤية هذه القيم ، اضغط مرتين خاصية الـ VertScrollBar أو الـ HorzScrollBar للـ form . تحدد قيمة الزيادة عدد الـ unit ويقوم thumb box بتحريكها عندما تضغط أزرار اسهم scroll bar . يحدد الـ Margin كم يبعد موقع scroll bar من حافة الـ form ويمثل الـ Position الموضع الحالي لـ thumb box .

● إذا تم تحديد نافذتين أو أكثر ليستقرن على القمة ، فإنهما يتحولان إلى نوافذ واقعة فوق بعضها البعض . فلا شيء باستطاعتك أن تفعله حيال هذا . لا يمكن أن تبقى كل نافذة على القمة طوال الوقت .

● يمكنك إدخال template من الـ Object Repository بثلاث طرق : الـ Copy أو Inherit أو Use . وهذه الخيارات يتم عرضها كالـ Radio عندما تختار الـ File/New... لفتح New Items dialog ، والذي يضع قائمة بمحتويات الـ Object Repository . وكل الخيارات الثلاثة لا تكون متوفرة لكل أنواع templates .

• يقوم خيار ال Copy بنسخ ال template فى تطبيقك إن حفظ هذه النسخ إذن ينشئ نسخة منفصلة للبيود، والتغييرات المستقبلية للملفات الأصلية ل template لا تنعكس على تطبيقك.

• ينشئ خيار ال Inherit، class مشتقه جديدة للاستخدام فى تطبيقك ويمكنك اضافة برمجة جديدة لها. واية تغييرات فى ملفات ال template الأصلية يشتمل عليه تطبيقك تلقائياً عندما تعيد عملية ال compile. وهذا يجعل خيار ال Inherit أكثر الطرق إفادة ومرونة فى استخدام ال template - على سبيل المثال لتحديث تطبيق، تقوم بتعديل ملفات ال template الاصلية ببساطة، ثم تجرى عملية ال compile جميع التطبيقات التى تستخدم هذه ال template. ولا يجب عليك إعادة إضافة ال template.

• والخيار الثالث والأخير، هو خيار ال Use، يستخدم ملفات ال template الأصلية مباشرة. على سبيل المثال، يتم فتح ال source code ل form مباشرة فى نافذة code editor وآية تغييرات تقوم بها فى ال form فى تطبيقك يتم حفظها فى ملفات اصلية لل template. لاستخدم هذا الخيار اذا كنت تريد الحفاظ على التصميم الاصلى لل template استخدمه فقط عندما تريد التشارك فى ال template مع تطبيقات متعددة. وهذا يعتبر الخيار الأقل مرونة ولكنه مفيد فى ال template مثل ال data modules.

## المشروعات التى يمكنك تجربتها

(٣-١): قم بتصميم وإضافة ايقونة نظام لتطبيق ال BitView استخدم ال Windows Paintbrush لل Image Editor الخاص ب Delphi's لإنشاء ملف ايقونة له امتداد اسم ملف ico. أو، انسخ ملف ايقونة موجود بالفعل من شبكة، أو برنامج مجاني للعرض فقط، أو قرص حقل عام. اختر MainForm، واضغط مرتين خاصية ال Icon التابعة له. استخدم ال dialog الناتج لفتح ملف الايقونة الخاص بك. يقوم Delphi بنسخ صورة الايقونة فى ملف ال exe. التام- لان يجب عليك توزيع ملف ico. لمستخدمى برنامجك.

### الباب الثالث : معلومات أولية حول الـ Forms

(٣-٢): استخدم الـ Delphi's AboutBox form الخاصة بـ Delphi من الـ template . اضع حق الطبع الخاص بك ، اسم الشركة ، والمعلومات - العامة الأخرى . قد تريد أيضاً أن تجعل الـ AboutBox مخصص ، لعرض اللوجو الخاص بالشركة (استخدم Image component) .

(٣-٣): قم بإنشاء الـ form template للحصول على إجابات بـ Yes أو No للمقترحات المختلفة . على سبيل المثال ، قد يعرض dialog سؤالاً وله أزرار Yes و No . إذا تعثرت ، إرجع لمشروع الـ YesNo على القرص المدمج الخاص بهذا الكتاب . افتح المشروع ، واتبع التعليمات الموجودة بهذا الباب لإضافة الـ YesNoDlg form داخل الـ Object Repository . إجعل اسم الـ form unit بـ Yesno.pas .

(٣-٤): متقدم . اكتب الـ text-file lister الخاص بك باستخدام الـ BitView2 كمرشد لك في مكان الـ Image ، استخدم Memo . لا يجب عليك إضافة الـ ScrollBar لتنفيذ الـ form ، رغم أنه باستطاعتك أن تفعل ذلك إذا أردت . رغم ذلك ، من الأسهل أن تختار الـ Memo الخاص بالـ form وتحدد خاصية الـ scroll bars التابعة له بـ ssBoth . إذا تعثرت إرجع إلى مشروع الـ ListText على القرص المدمج لهذا الكتاب .

(٣-٥): استخدم الـ Splitter لإنشاء two view لبرنامج قائمة الملف الخاص بك من مشروع (٣-٤) .

### ملخص:

- كل مشروع له form object واحد على الأقل يمثل النافذة الرئيسية للبرنامج . ولكن ، قد يكون للمشروعات forms متعددة .
- الـ form تعتبر component ، ولكنها لا تظهر في component palette . قم بإنشاء form باستخدام مشروع جديد أو باختيار FileNew . Form....

• كل form لها unit module مقابل يضع فى قائمة ال forms الخاص للبرنامج . يجب ان تختلف خاصية ال Name لل form عن اسم ال unit file (فكرة : لتجعل ملفات واسماء ال form الخاصة بك صحيحة استخدم Name مثل EntryForm واحفظ ال source code unit فى ملف يسمى (Entry.pas) .

• تعتبر ال form اكثر من مجرد تمثيل لنافذة البرنامج . ان ال forms لها ايضاً events وخصائص يمكنك برمجتها لاختيار مواصفات النافذة ولأداء اعمال أخرى .

• ان ال forms و templates app. تفيد فى إنشاء شاشات عامة ، انماط نموذجية ، و demos اختر امر ال FileNew..... الاختيار من بين template العديد المتوفرة ، أو يمكنك إنشاء ال template الخاصة بك وإضافتها لل Object Repository الخاص بـ Delphi .

• يمكنك تحديد أى form كنافذة رئيسية للتطبيق على سبيل المثال ، يمكنك استخدام dialog box للحصول على واجهة تطبيق لتطبيق نظيف وبسيط لا يتطلب menu bar .

• ال OnCloseQuery يؤكد أو يلغى إغلاق نافذة . استخدم هذا ال event للمساعدة فى منع فقد البيانات .

• تقوم ال Scroll bar بتشغيل نوافذ لعرض مزيداً من المعلومات اكثر مما يتناسب داخل حدود النافذة . من السهل إضافة Scroll bar سرية للنوافذ ، ولكن فى التطبيقات التامة ، يجب ان تقوم بعدة تعديلات لل Scroll bar الخاصة بنوافذك كما هو موضح فى هذا الباب .

• قم بتعين خاصية FormStyle لل form بـ fsStayOnTop لتجعل هذه النافذة فوق نوافذ التطبيق الأخرى . يمكنك تعيين هذه القيمة فى ال Object Inspector ، أو اثناء وقت التشغيل .

• يمكن للمشروع ان يكون له forms متعددة كما يظهر برنامج ال Splashin . يقوم Delphi عادة بإنشاء ال objects فى وقت التشغيل ، وهى حقيقة قد يكون لها آثار على الذاكرة وال resources الأخرى فى التطبيقات الكبيرة . لإنشاء ال form object فى ظل سيطرة البرنامج ، استخدم ال Project

### الباب الثالث : معلومات أولية حول الـ Forms

Manager لتحديد المشروع ، اضغط زر الفأرة الأيمن ، واختر أمر الـ Options . فى الـ Forms page ، قم بإزالة الـ form من قائمة form . لم يعد يتم إنشاء الـ form object تلقائياً .

● يمكن للـ data modules ان تحتفظ بـ objects مثل الـ MainMenu . وهذا ينقل الـ component خارجاً ويساعد على تنظيم أفضل للبرنامج . لاستخدام الـ data modules ، أدخل اسم الـ unit فى الـ uses directive فى قطاع الـ implementation للـ form unit يمكن تخزين الـ Data modules فى الـ Object Repository .

● استخدم الـ Splitter لإنشاء نوافذ متعددة يمكن للمستخدمين إعادة تحديد حجمها فى وقت التشغيل بالضغط والسحب الـ Splitter bar .

فى الباب القادم ، سوف تبدأ فى تعلم كيفية استخدام الـ Delphi components and forms لتصميم واجهات تطبيق جذابة للمستخدم . سوف تعرف أيضاً كيف ترمج اثنين من أهم أجهزة الإدخال فى الحاسوب : لوحة المفاتيح والفأرة



# الجزء الثاني

## واجهة تطبيق المستخدم

### محتويات هذا الجزء:

- الباب الرابع: برمجة لوحة المفاتيح والفأرة
- الباب الخامس: constructing Mouses
- الباب السادس: إلحاق الـ Attaching Buttons and Check Boxes
- الباب السابع: إنشاء Coolbars، Toolbars، و Status Panels
- الباب الثامن: Making Lists
- الباب التاسع: العمل مع Single Line Strings
- الباب العاشر: العمل مع Multiple-Line Text
- الباب الحادي عشر: Navigating Directories and Files
- الباب الثاني عشر: Communicating with Dialog Boxes

إن إنشاء واجهة تطبيق كفاء للمستخدم هو أمر غير يسير. بالطبع، يمكن لغالبية الناس مع وجود معمل مناسب أن يروا buttons، check boxes، حقول الإدخال، والقوائم ليذكروا شيئاً عملاقاً يشبه تطبيق الـ Windows. ولكن هذا يتطلب عمل شاق وتخطيط حذر لإنشاء واجهات تطبيق جيدة تجذب المستخدمين بدلاً من أن تفرزهم.

إن الأبواب الموجودة في هذا الجزء تغطي تقنيات و Delphi Components لتجميع واجهات التطبيق الصديقة والعملية. سوف تعرف أدق التفاصيل عن إنشاء واجهات تطبيق جرافيكية للمستخدم، واستخدام Component object مثل القوائم، الأزرار، Toolbars، status panels، lists، scrollbars، dialog boxes، بالإضافة إلى إجراء جسم التطبيق الأخرى.





# الباب الرابع

## برمجة لوحة المفاتيح والفأرة

### محتويات الباب:

• حول الجزئين الثانى والثالث

• على لوحة المفاتيح

• مصائد الفأرة

يوماً ما، ولتأمل ان يكون قريباً، سيكون التحدث مع الحاسوب أمراً طبيعياً كالثرثرة مع الجيران. ولكن حتى يتم عمل التحسينات الكبرى لبرمجيات وآلات التعرف الصوتى، سوف تبقى لوحة المفاتيح والفأرة أداتى الادخال الاسائتين للتخاطب مع تطبيقات البرمجيات. ان الحاسبات يمكنها ان تتحدث بشكل جيد ولكنها لا تعرف كيف تسمع.

كما يعلم المطورين الخبراء، ان امكانات الادخال الخاصة بتطبيق تعرف حرفياً سهولة استخدامه. بالضغط على مفتاح أو أمر فأرة واحد غير ضرورى بامكانه ان يزعج اكثر المستخدمين خبوة والذين لا يرغبون فى كتابة أو ضغط اكثر من الضرورة القصوى. ولان سهولة الادخال تعتبر امر حتمى لنجاح البرنامج، فان مواد استعمال الفأرة ولوحة المفاتيح تعتبر اماكن مناسبة لبداية التحقيقات للجزء الثانى فى تطوير واجهات تطبيق مستخدم التطبيق مع ال Visual Component Library الخاصة بـ Delphi.

### حول الجزئين الثانى والثالث:

ان الجزء الثانى من هذا الكتاب يغطى اساليب تصميم واجهة تطبيق المستخدم باستخدام ال Delphi components مثل القوائم، الازرار، وال objects الاخرى

المتصلة بواجهة التطبيق ويشرح الجزء الثالث موضوعات مختصة بالتطبيق مثل الرسوم الجرافيكية والتشارك فى البيانات، ويغطى objects متقدمة فى ال Visual Component Library. لمعرفة غالبية هذه المعلومات، يجب ان تعرف كيف تؤدى المهام التالية :

- ابدأ تطبيق Delphi جديد.
  - عين خصائص ال Name وال Caption ل form.
  - أضف component object على ال form.
  - أضف form جديدة الى التطبيق.
  - قم بتعديل خصائص ال component objects وال form.
  - قم بإنشاء procedure لل component ولل form.
- إذا كنت لا تعرف أى من البنود السابقة، فكلها معرفة فى الجزء الأول، إقرأ الابواب الثلاثة السابقة وأكمل الى النهاية البرامج المعروضة خطوة بخطوة. تأكد من انك تعى كيفية عمل المهام السابقة قبل الاستكمال.

### : Components covered

ان الابواب الموجودة فى الجزئين الثانى والثالث تبدأ بقائمة ووصف مختصر لل covered components وال classes المرتبطة بها. وتشير غالبية الابواب أيضاً ل component اخرى والبرامج التعليمية والأمثلة وقد تم وضع كشف بكافة ال component التى سوف التعامل معها. ويستخدم هذا الباب components متعددة، ولكنه يركز على ال TForm class الموضحة فيما يلى :

- Form - تعتبر نوافذ التطبيق ل objects ال Form component والذى تم برمجته فى TForm class. ويحتوى ال Form object على components اخرى مثل ازرار، edit boxes، وعناصر اخرى تنشئ واجهة التطبيق لمستخدم التطبيق.

**ملحوظة:** يبدأ أى مسمى لل component's Pascal class بحرف ال T، مما يعنى نوع البيانات. على سبيل المثال، TButton هو اسم Button component class. فى هذا الكتاب، إننى استخدم اسماء

Note

## الباب الرابع : برمجة لوحة المفاتيح والفأرة

class فقط عند اللزوم - على سبيل المثال ، يتم تعريف متغير وذلك تحديد ال component class له .

### البرامج التعليمية وجدول الخصائص:

لإختصار التعليمات الموضحة خطوة بخطوة ولتفادي تكرار الشرح الواضح (مثل كيفية حفظ ال units وملفات المشروع) ، فإن البرامج التعليمية العملية التالية تعتبر مختصرة أكثر منها في الجزء الأول . بالإضافة الى ذلك ، تضع بعض البرامج التعليمية كشف يقيم الخصائص في جدول . يظهر جدول (٤-١) تنسيق جدول الخاصية المستخدم خلال باقى الكتاب .

جدول (٤-١) : عينة لجدول الخصائص

Component	Name	Property	Value
Form	MainForm	Caption	Widow Demo
Label	XCoordLabel	Caption	X Coordinate
		Alignment	taRightJustify
		Font.Name	Arial
		Font.Size	24
Edit	Edit1	Text	

إن السطر الأول من جدول الخاصية يضع قائمة تحتوى على أسماء لل component objects على سبيل المثال ، يشير الجدول (٤-١) الى أنه يجب عليك تغيير خاصية ال Name لل form الأصلية للبرنامج ليكون MainForm ، وحدد ال Caption الخاص به بـ Window Demo . يجب عندئذ إدخال ال component الأخرى المدرجة فى القائمة داخل نافذة ال MainForm .

**على القرص المدمج:** كل ال main-window فى هذا الكتاب تسمى

MainForm ، الا اذا تم الاشارة الى غير ذلك . على القرص المدمج ، ان

MainForm unit module ، وهى Main.pas ، مخزنة فى دليل

فرعى يتلائم مع اسم المشروع مثل ال BitView و OnTop . و form modules

الأخرى مسماة بطريقة مشابهة على سبيل المثال ، NextForm unit module ،

سوف تسمى Next.pas .



والسطر الثانى من الجدول (٤-١) يشير الى انه يجب عليك إضافة Label component object فى ال MainForm وتغيير ال Name الخاص به ليصبح XCoordinate . قم بتحديد ال Caption الخاص بهذا ال object بـ XCoordLabel ، وقيم بتغيير خصائص ال Alignment وال Font الى القيم المشار إليها . بالنسبة لحجم ومكان ال object ، إرجع الى ال objects الموجودة بالكتاب ، والى الملفات المتوفرة .

تفصل النقطة بين الخاصية والقيمة الفرعية الخاصة بها . على سبيل المثال ، يشير عمود ال Property فى الجدول (٤-١) الى انه يجب عليك تحديد القيمة الفرعية Name لخاصية ال Font بـ Arial وقيمتها الفرعية Size بـ ٢٤ نقطة .

والسطر الأخير من الجدول (٤-١) يخبرك بان تضيف Edit component object الى ال MainForm ، ولكن ان تترك ال Name الخاص بال object محدد بالقيمة الافتراضية ، وهى Edit1 . والعمود الاخير الحاوى يشير الى انه يجب عليك إخلاء خاصية ال Text لل object بإبراز وحذف هذه القيمة .

ان جداول الخاصية لا تتضمن التحديدات الواضحة أو الغير هامة مثل قيم طول وعرض النافذة . ولا يذكر الجدول ايضاً مظاهر النافذة مثل ال Bevels المستخدمة فى مشروع ال KeyInfo لتنظيم عرض النافذة [راجع الشكل (٤-١)] . إننى سوف اشرح كثير من هذه الاشياء الإضافية خلال الكتاب ، ولكننى لا اريد إهدار مساحة فى شرح اشياء واضحة على حساب معلومات اكثر اهمية .

طبقاً للعرف ، تنتهى غالبية خصائص ال Name لل component بـ component identifier . على سبيل المثال ، انك تعرف ان XCoordLabel الموجودة فى جدول (٤-١) هى Label component object . والاستثناء المعتاد لهذه القاعدة هو اسم افتراضى حسب النظام مثل Edit1 ، الذى يعنيه Delphi .

### على لوحة المفاتيح:

يمكن لتطبيقات Delphi ان تستخدم two methods اساسين لتلقى مدخلات لوحة المفاتيح . ال method الاول والاسهل هو ان يستخدم ال component object ، مثل ال Edit ، الذى يستجيب بصورة تلقائية للضغط على المفاتيح لأغراض اكثر شمولاً ، يمكنك إنشاء procedures فى form تتعامل

## الباب الرابع : برمجة لوحة المفاتيح والفأرة

مع أى مجموعة من ثلاث events (موجودون فى قائمة ال form object فى نافذة ال Object Inspector):

● **OnKeyDown** : يستدعى عندما تضغط أى مفتاح، بما فى ذلك ال function key والمفاتيح الخاصة مثل Shif ، Alt ، و Ctrl .

● **OnKeyPress** : يستدعى عندما تضغط مفتاح إنتاج الرمز ASCII ، بما فى ذلك مفاتيح التحكم .

● **OnKeyUp** : يستدعى عندما تترك أى مفتاح .

من هذه events يتلقى parameter واحد على الأقل، يسمى Key، والذي يمثل مفتاح قد تم الضغط عليه، فى ال OnKeyDown وال OnKeyUp، ويحدد نوع ال Key بـ unsigned Word (أى، موجب فقط) والذي يمثل . virtual key code لل Windows . فى ال OnKeyPress، يكون لل Key قيمة ال Char والتي تمثل رمز ال ASCII . رغم ان كلا المتغيرين يسمى Key، الا انهما يمثلان معلومات مختلفة من لوحة المفاتيح . لكل رموز ال ASCII، virtual key code مقابلة لها، ولكن العكس ليس صحيح - فهناك كثير من ال virtual key ليس لها نظراء من ال ASCII .

يعبر ال Windows عن keystrokes لكل virtual key code برمز يسبقه وهو vk\_ . على سبيل المثال، يعتبر ال vk\_alt code ال virtual key لل Alt . لمزيد من المعلومات حول virtual key code، إبحث عن دالات ال VkKeyScan وال VkKeyScanEx فى ملف ال Microsoft Win32.hlp المتوفر مع Delphi . اذا لم يكن لديك هذا الملف أو لم تتمكن من وضع الابعاد (يبدو انها مفقودة فى بعض الطباعات)، إبحث عن ال Vk فى ال Windows.pas الموجود فى دليل ال Source\Win الخاص بـ Delphi .

### الاستجابة لل event الخاصة بلوحة المفاتيح:

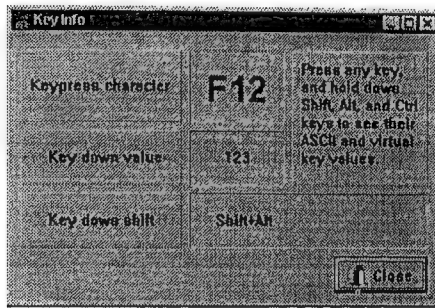
يظهر التطبيق، KeyInfo، كيفية برمجة ال event الخاصة بلوحة المفاتيح الثلاث التابعين لل form . يوضح الشكل (٤-١) ظهور ال KeyInfo . قم بوضع وتشغيل هذا البرنامج بتحميل ملف مشروعه من دليل ال KeyInfo ويضغط F9 .

اضغط مفاتيح متنوعة، بما في ذلك الوظيفة، Ctrl، Shift، و Alt - ولاحظ القيم المعروضة إلى اليمين من بطاقات النافذة:

• **Keypress character**: يوضح هذا ال label رموز ال ASCII. وهناك مربع خالي معروض للرموز التي ليس لها رموز مرئية.

• **Key down value**: يوضح هذا ال label القيمة العددية للمفاتيح. عندما تضغط مجموعة مفاتيح متعددة مثل Shift+Alt+A، ويوضح هذا المربع قيمة آخر مفتاح تم ضغطه (فهو A في هذا المثال).

• **Key down shift**: يوضح هذا ال label حالة الانتقال الحالية. على سبيل المثال، عندما تضغط مفتاحي Alt و Ctrl معاً، يعرض البرنامج Alt+Ctrl، وتوضح قيمة ال Key down value القيمة العددية للمفتاح الذي تم ضغطه آخراً (وهو Ctrl في هذه الحالة).



شكل (٤-١): يعرض ال KeyInfo ال events الخاصة بلوحة المفاتيح

رغم ان ال KeyInfo يظهر معلومات لغالبية keypresses، يستمر البرنامج في الاستجابة لأوامر لوحة المفاتيح المعيارية. على سبيل المثال، ان ضغط Enter يختار زر ال Close وينهى البرنامج. وضغط ال F10 يختار ايقونة قائمة النظام، التي يمكنك بعدها ان تضغط قضيب المسافة أو Enter لفتح القائمة، أو اضغط Esc للعودة للتشغيل العادي.

### إنشاء ال KeyInfo :

إتبع الخطوات المرقمة في هذا الفصل لإعادة إنشاء التطبيق KeyInfo. يوضح الجدول (٤-٢) قيم خصائص ال form وال component الهامة.

جدول (٢-٤) : خصائص الـ KeyInfo

Component	Name	Property	Value
Form	MainForm	Caption	Key Info
Label	CharLabel	Caption	
		Alignment	taCenter
		AutoSize	False
		Font:Name	Arial
		Font:Size	24
		Font:Style	[fsBold]
Label	ValueLabel	Alignment	taCenter
		AutoSize	False
Label	ShiftLabel	Alignment	taLeftJustify
		AutoSize	False

نفترض بالتعليمات التالية انك قد قمت بانشاء مشروع جديد، وغيّرت الـ Name لـ form النافذة الرئيسية ليصبح MainForm، وقمت بحفظ الـ unit كـ module Main والمشروع كـ KeyInfo لإنشاء التطبيق KeyInfo:

١- أضف الـ Bevels والـ Labels على الـ form لتنظيم النافذة كما هو موضح في شكل (١-٤). ولا يهم الموضع الدقيق لهذه البنود. استخدام الخاصية الافتراضية Names في Delphi لهم جميعاً.

٢- بإضافة الـ BitBtn من الـ Additional. قم بتغيير الـ Kind الخاص به ليصبح bkClose والـ Name الخاص به ليكون CloseBitBtn.

٣- قم بإضافة ثلاث Label's في منتصف لـ Bevels المفرغة كما هو موضح في شكل (١-٤). إجعل اسماءها CharLabel، ValueLabel، و ShiftLabel على التوالي، واحذف الـ Captions الخاصة بهم. إرجع الى جدول (٢-٤) للحصول على قيم خصائص أخرى. قم بتعديل خصائص الـ Font للتوصل الى عرض مناسب - على سبيل المثال، قد تريد ان تجعل الـ CharLabel Font كبير وسميك.

٤- اختر MainForm، وقم بانشاء الـ OnKeyDown، OnKeyPress، و OnKeyUp. قم بادخال البرمجة الموضحة في القائمة (١-٤) لهذه الـ procedure الثلاثة. سوف اشرح كيف تعمل هذه العبارات بعد القائمة.



٥- اضغط F9 لإجراء عمليتي ال compile و ال link ومن ثم تشغيل

المنتج .

على القرص المدمج؛ يمكنك ان تجد source code كاملة لـ KeyInfo  
على القرص المدمج المرفق بهذا الكتاب في دليل ال KeyInfo's؛ توضح  
القائمة (٤-١) ملف Main.pas الخاص به.



القائمة (٤-١): Keyinfo\Main.pasunit Main;

interface

uses

Windows, SysUtils, Messages, Classes, Graphics, Controls,  
Forms, Dialogs, Buttons, StdCtrls, ExtCtrls;

type

TMainForm = class(TForm)

Label2: TLabel;

CharLabel: TLabel;

Label3: TLabel;

Label4: TLabel;

ValueLabel: TLabel;

ShiftLabel: TLabel;

Bevel1: TBevel;

Bevel2: TBevel;

Bevel3: TBevel;

Label1: TLabel;

Bevel4: TBevel;

Bevel5: TBevel;

Bevel6: TBevel;

Bevel7: TBevel;

CloseButton: TBitBtn;

procedure FormKeyDown(Sender: TObject; var Key:  
Word;

```
Shift: TShiftState);
    procedure FormKeyPress(Sender: TObject; var Key:
    Char);
    procedure FormKeyUp(Sender: TObject; var Key: Word;
    Shift: TShiftState);
    private
    { Private declarations }
    public
    { Public declarations }
    end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

const
    ctrl_A = 1;    { ASCII value for Ctrl+A }
    ctrl_Z = 26;   { ASCII value for Ctrl+Z }

    FunctionKeys: array [vk_f1 .. vk_f12] of string[3] =
        ('F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8',
         'F9', 'F10', 'F11', 'F12');

    procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;
    Shift: TShiftState);
    var
        s: string;
    begin
        {- Show integer Key value }
```

```

ValueLabel.Caption := IntToStr(Key);
{- Show Key shift state }
s := "";
if ssShift in Shift then s := s + 'Shift+';
if ssAlt in Shift then s := s + 'Alt+';
if ssCtrl in Shift then s := s + 'Ctrl+';
if Length(s) > 0 then
  Delete(s, Length(s), 1); { Delete final '+' }
  ShiftLabel.Caption := s;
{- Do function key labels }
if Key in [vk_f1 .. vk_f12] then
  CharLabel.Caption := FunctionKeys[Key]
else
  CharLabel.Caption := ""; { Erase old character label }
{- Disable Spacebar to prevent selecting Close button }
if Key = vk_space then
  Key := 0;
end;

```

```

procedure TMainForm.FormKeyPress(Sender: TObject; var
Key: Char);
begin
if Ord(Key) in [ctrl_A .. ctrl_Z] then
  CharLabel.Caption := Chr(Ord(Key) + Ord('A') - 1)
else
  CharLabel.Caption := Key;
  ValueLabel.Caption := IntToStr(Ord(Key));
end;

```

```

procedure TMainForm.FormKeyUp(Sender: TObject; var Key: Word;
Shift: TShiftState);
begin

```

```
{ - Erase the three labels when user releases key(s) }
CharLabel.Caption := "";
ValueLabel.Caption := "";
ShiftLabel.Caption := "";
end;

end.
```

### داخل KeyInfo :

ان إثنين من الثوابت، وهما ctrl\_A و ctrl\_Z فى قطاع implementation ال Main يقرران قيم ال ASCII لمفتاحى ال Ctrl+A وال Ctrl+Z. يستخدم البرنامج هذه القيم لتحديد مدى مفاتيح التحكم بحيث، عندما تضغط Ctrl+Q، مثلاً، يوضح ال label رمز ال Keypress لهذه القيمة.

ويحدد الثابت الثالث function key strings مثل F1 و F2، أيضاً ليظهر فى ال label رمز ال Keypress. ومجموعة الثوابت المكتوبة هذه تأتى تلقائياً فى البداية قبل القيم الموضحة فى الاقواس، وفى هذه الحالة، فهى مفهومة بـ virtual key codes، vk\_f1 الى vk\_f12. وباعطاء variable string اسم ال FunctionS، تحدد العبارة التالية ال string مساوياً لـ F3:

```
FunctionS := FunctionKeys[vk_f3]; { Assign F3 key to FunctionS }
```

وبرنامج ال event handler الخاص بلوحة المفاتيح، وهو FormKeyDown، يتلقى قيمة ال Key وهى تساوى virtual key code لمفتاح مضغوط. أولاً، يعين ال procedure قيمة Key الخاصية ال Caption التابعة للـ ValueLabel object من اجل عرض هذه القيمة تحت ال Label ال Key ال down value فى النافذة. ولكن لان ال Captain يعتبر string، فلا يمكنك تعيين Key مباشرة له. فهذا لن يعمل:

```
ValueLabel.Caption := Key; { ??? }
```

**ملاحظة:** خلال هذا الكتاب، استخدمت ثلاث علامات استفهام في اقواس تعليق، {??}، لأشير الى عمل قابل للتسائل، أو عبارة خاطئة يجب ان تتلافى استخدامها في برامجك.

Note

بدلاً من ذلك، يجب ان تحول ال Key الى قيمة string فى هذه الحالة، يمكنك ان تفعل هذا بان تمرر ال Key الى ال IntToStr الخاص ب Delphi، والتي تقبل الأنواع المعرفة ب Word وتعيد ال string:

```
ValueLabel.Caption := IntToStr(Key);
```

والعبارة التالية فى ال FormKeyDown تعين ال null string لل s. وهذا ينشئ ال string لعبارة ال if القادمة:

```
s := "";
```

**فكرة:** لتحويل ال string الى قيمة عدد صحيح، استخدم ال

StrToInt.

لإنشاء ال null string، اكتب علامتى تنصيص فردية بلا مسافة بينهما. ان تعيين ال null string لتغيير ال string من خلال المعامل =: يحو ال string characters.

وتبنى ثلاث عبارات ال if ال string s لتوضيح قيم ال key المجتمعة مثل Shift+Ctrl. ويستخدم العبارات ال parameter ال Shift، من نوع ال TShiftState، وقد تم تمريرها لل parameter. يعرف ال Delphi نوع بيانات ال TShiftState كالآتى:

```
TShiftState =
```

```
set of (ssShift, ssAlt, ssCtrl, ssRight, ssLeft,  
ssMiddle, ssDouble);
```

ومتغير مثل Shift - مجموعة من قيم ال TShiftState - يمكن ان يحمل قيم صفر، واحد، أو أكثر. على سبيل المثال، يمكنك تعيين هذه المجموعة ل Shift لتمثيل مفتاحى ال Shift و ال Ctrl:

```
Shift := [ssShift, ssCtrl];
```

## الباب الرابع : برمجة لوحة المفاتيح والفأرة

يعرف Pascal ما يسمى set بـ single bits ، لذا تعتبر هذه الطريقة فعالة لتخزين عناصر متعددة في مساحة صغيرة للغاية .

وتحدد سلسلة من عبارات الـ if أى من القيم الممكنة (ان وجد) تعتبر فى الـ Shift set . اختبر أولى هذه العبارات :

```
if ssShift in Shift then s := s + 'Shift+';
```

هذا يعنى انه اذا كانت القيمة ssShift فى مجموعة الـ Shift ، فإن البرنامج يضم string + Shift الى نهاية المتغير s . وعبارات مشابهة . تفحص قيم مجموعة اخرى ، وتضم string إضافية الى الـ s .

بعد تكوين الـ string ، يكون المتغير s إما خالياً ، أو يصنع كشفافاً باسماء مفاتيح خاصة متنوعة ، تفصلها علامة زائد . اذا لم يكون الـ string خالياً ، فإنه ينتهى بعلامة زائد إضافية ، والتي تحذفها هذه العبارة :

```
if Length(s) > 0 then  
Delete(s, Length(s), 1);
```

ويقوم Length procedure فى الـ Pascal بتحديد الطول لأى string . اذا كان هذا الطول اكبر من صفر ، تستدعى الـ if هنا Delete لحذف الرمز الأخير . للحذف ، قم بتمرير الـ string فى الاتجاه الذى يبدأ منه الحذف ، وعدد الرموز التى ستحذفها من هذا الموضع ولان الرمز الأول فى الـ string يوجد . فى الموضع -1 بمعنى آخر ، فى فهرس الـ string -1 فإن تمرير طول الـ string للـ parameter الثانى لـ Delete يحذف الرمز الأخير فى الـ string .

لعرض الـ string ، يعينه البرنامج لخاصية الـ Caption للـ ShiftLabel :  
ShiftLabel.Caption := s;

للتعامل مع function keys ، تختبر عبارة if أخرى اذا ما كان الـ Key فى مجموعة . virtual key codes من الـ vk\_f1 الى vk\_f12 . اذا كان كذلك ، يعين البرنامج احد الـ strings الثابتة للـ unit من مجموعة الـ FunctionKeys للـ Caption الخاص بـ CharLabel . وان لم يكن كذلك ، يحدد البرنامج هذا الحقل بالـ null string ، بمسح أى نص موضح فى هذه المساحة .

ويعين ال FormKeyDown أيضاً ل null string ل CharLabel.Caption  
لمسح أى رمز معروض فى هذا المجال :

CharLabel.Caption := ";

واخيراً، فى ال FormKeyDown، تختبر عبارة if إذا ما كان ال Key  
يساوى مسافة ال vk\_space. لقد اضيفت هذه العبارة لأن ضغط قضيب المسافة  
يختار ال control object الحالى - فى هذه الحالة، CloseButton. لاحظ ان  
متغير ال Key فى رأس (أى السطر الأول من) ال FormKeyDown مسبق بـ  
var. هذا يعنى أن parameter متغير وتعين قيمة له يمرر هذه القيمة لإستدعاء ال  
FormKeyDown. لذا، فإن تحديد ال Key مساوياً لصفر ليقف عمل هذا المفتاح  
(لأن الصفر لا يمثل مفتاحاً بعينه):

if Key = vk\_space then

Key := 0; { Disable SpaceBar }

ال two procedures التالين يسهل فهمهما نسبياً. يعين ال FormKeyUp  
assigns null لخصائص ال Caption فى ال CharLabel، ValueLabel، و  
ShiftLabel؛ وهذا يسمح تلك ال labels عندما تترك المفتاح.

ويتلقى ال FormKeyPress قيمة ال Key والتى تساوى رمز ASCII. اذا كانت  
قيمة ال Key الموجودة فى مجموعة القيم المحددة بواسطة مدى ال control-code،  
من ال ctrl\_A الى ctrl\_Z، عندئذ يحدد البرنامج ال Caption الخاص بـ  
CharLabel بـ '^A'، '^B'، وما الى ذلك. (وغالبا ما تستخدم علامة ^ كرمز لمفتاح  
تحكم). وهذا ضرورياً لأن control code ليس لها رمز مرئى مرتبط بها.

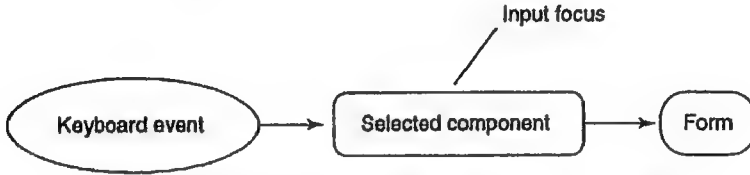
وتعالج عبارة ال noncontrol characters else ببساطة بتعيين ال Key  
مباشرة الى ال Caption الخاص بـ CharLabel. لاحظ أنه بإمكانك تعيين متغير  
Char مثل Key ل string مثل ال Caption واحدة من المساحات القليلة التى يكسر  
فيها Pascal قواعده القوية.

وأخيراً، يحدد البرنامج حقل ال Caption الخاص بال ValueLabel  
بمساواته لل string مع القيمة الخاصة بال Key. عندما تضغط Q، مثلاً، تعين هذه  
العبارة string ال 81 لل Caption.

## مفهوم Keypresses:

يستخدم تطبيق الـ KeyInfo ، event handler ، في الـ form لتستجيب لعملية الضغط على المفاتيح . وتستجيب أيضاً كثير من الـ components لنشاط لوحة المفاتيح ، ومن الضروري عادة أن تحدد اذا كان الـ component أو الـ form يجب أن يتلقى الـ event الخاص بلوحة المفاتيح .

يستخدم الـ Windows ، الـ input focus لتحديد أين يرسل الـ event الخاصة بلوحة المفاتيح . على سبيل المثال ، الـ component الذى يتم إختياره حالياً في الـ form له . input focus ولذا ، فإنه يتلقى كل الـ event الخاصة بلوحة المفاتيح . اذا لم يتعامل الـ component مع الـ events ، مثلاً ، لأنه لم يتم برمجته على استخدام مفتاح معين مثل الـ Esc أو Ctrl ، فإن هذا الـ events يتم تمريره للـ component الذى يتم التعامل معه والذي هو فى الغالب الـ form .

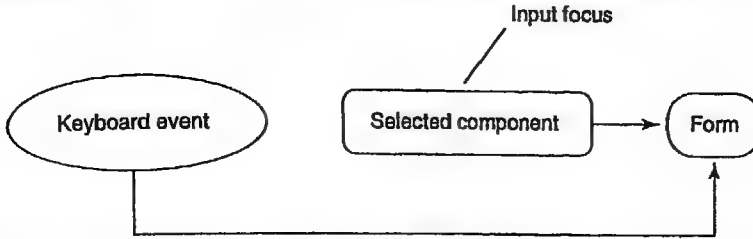


شكل (٤-٢) : تذهب الـ events الخاصة بلوحة المفاتيح  
فى العادة الى الـ component المختار ثم الى الـ form

لإرسال الـ events الخاصة بلوحة المفاتيح الى الـ form قبل أن يتلقى الـ component المختار هؤلاء الـ events ، حدد خاصية الـ KeyPreview للـ form بـ True . على سبيل المثال ، قد تقوم بهذا لتعديل مفاتيح معينة أو لمنع المستخدمين من كتابة رموز غير مطلوبة مثل الأرقام أو علامات الترقيم . يوضح الشكل (٤-٣) العلاقة بين الـ component ، والـ form ، و input focus عندما يكون الـ KeyPreview فى الـ True .

وتوضح تجربة بسيطة قيمة خاصية الـ KeyPreview ، والتي يمكنك استخدامها لتحديد المدخلات فى الـ ext-editing component . إبدأ مشروع جديد ، وإتبع الخطوات التالية :





شكل (٣-٤)، حدد الـ KeyPreview بـ True لتجد الـ events الخاصة بلوحة المفاتيح في الـ form قبل أن تستلم الـ component المختار هؤلاء الـ events

١- أدخل الـ Edit في الـ form إنك لا تحتاج الى تغيير الـ Name الخاص به أو أى من الخصائص الأخرى.

٢- قم بتغيير خاصية الـ KeyPreview للـ form من False الى True بالضغط مرتين على قيمتها.

٣- غير الى الـ Events page في الـ Object Inspector، واضغط مرتين الـ OnKeypress للـ form لإنشاء الـ event handler. تأكد من أنك فعلت هذا الـ Form1 object، وليس الـ Edit1. اكتب العبارة التالية بين البداية والنهاية لتحويل الحروف الصغيرة الى حروف كبيرة:

Key := Upcase(Key);

٤- اضغط F9 لتشغيل البرنامج. اذا عرض Delphi الـ file-save dialogs، اضغط الـ unit والمشروع تحت الاسماء الافتراضية في دليل مؤقت.

٥- اكتب بعض النص داخل Edit1. يقوم البرنامج تلقائياً بتحويل الحروف الصغيرة الى حروف كبيرة سواء قمت بضغط Caps Lock أو Shift، أو لم تضغطها.

٦- اترك البرنامج، حدد الـ KeyPreview بـ False، ثم اضغط F9 والتشغيل. لم يعد البرنامج يحول الرموز الى حروف كبيرة. وهذا يحدث لأنه أصبح الـ Edit1 له الآن input focus بدلاً من الـ form- ولذلك، يتلقى التحكم الـ events الخاصة بلوحة المفاتيح أولاً.

يعتبر الـ KeyPreview نافعاً أيضاً في تحديد أنواع الرموز التي يمكن للمستخدمين إدخالها. على سبيل المثال، لمنع المستخدمين من كتابة مفاتيح أرقام

## الباب الرابع : برمجة لوحة المفاتيح والفأرة

داخل ال Edit ، أضف هذه العبارة الى OnKeyPress الخاص بال form وحدد ال KeyPreview بـ True :

```
if Key in ['0' .. '9'] then  
  Key := Chr(0);
```

وتفحص عبارة if هذه اذا ما كان ال Key parameter الذى تم تمريره الى ال event handler موجود ضمن مجموعة الرموز من '0' الى '9' . اذا كان كذلك ، يعين السطر الثانى من العبارة قيمة الرمز صفر للـ Key . ولأن قيمة رمز ASCII صفر ليس لها معنى ، فهذا يمنع أى إدخال فى ال Edit معدل .

وتستخدم عبارة if السابقة بعض تقنيات Pascal القيمة التى قد تبدو جديدة بالنسبة لك . والتعبير '9' .. '0' يسمى مدى . إنه يتكون من قيمتين منفصلتين بواسطة نقطتين . ويترجم Pascal هذا التعبير كمساو لسلسلة الرموز :

'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

يمكنك أيضاً أن تؤلف نطاقاً ، من قيم أعداد صحيحة على سبيل المثال ، للمدى 8 .. 5 يساوى سلسلة القيم 5, 6, 7, 8 . تأكد من ان تفهم ان رموز ال ASCII وقيم الاعداد الصحيحة ليست متساوية لبعضها . ان النطاق '9' .. '0' يساوى سلسلة الرموز الرقمية ASCII . والنطاق 9 .. 0 (بدون علامات التنصيص) يساوى سلسلة قيم الاعداد الصحيحة من صفر الى تسعة . يمكنك فقط حشو القيم الترتيبية (المثلة بقيم اعداد صحيحة) فى مجموعة . لا يمكنك ، مثلاً ، ان تنشئ مجموعة من ال strings ، لان Pascal لا يستطيع تمثيل قيم ال string باجزاء فردية . ان ال array هو نوع البيانات المناسب لتجميع القيم المعقدة مثل strings ، class objects ، records .

وتوضح عبارة if السابقة ايضاً كيفية تعيين قيم ASCII خاصة لتغير من نوع ال Char . ولانه من المستحيل كتابة رمز بقيمة 0 (أو صفر) للـ ASCII ، فنحن فى إحتياج لاسلوب آخر لتعيين هذه القيمة للـ Key . ولان ال Key parameter يعد متغير من نوع Char ، يمكنك تعيين أى رمز له بعبارة مثل :

```
Key := 'Q';
```

رغم ذلك ، لا يمكنك تعيين قيمة ASCII مباشرة لتغير Char . فهذا لا يعمل :

Key := 81; { ??? }

ان القيمة 81 ليست رمزاً، رغم انها تعتبر كقيمة ASCII للـ Q. لا يمكنك تعيين الصفر مباشرة للـ Key:

Key := 0; { ??? }

ان Pascal صعب الإرضاء بالنسبة لمثل هذه التعيينات لان القيمة المعينة يجب ان تتطابق مع نوع المتغير. والحل هو تحويل القيمة الى نوع البيانات المناسب، والذي يمكنك فعله هنا باستخدام Chr function توضيح العبارات التالية ثلاث methods متساوية منطقياً لتعيين رمز الـ Q للـ Key:

Key := Chr(81); { Calls Pascal Chr to convert 81 to a Char }

Key := Char(81); { "Casts" the value 81 to the type Char }

Key := 'Q'; { Directly assigns the character Q to Key }

### إنشاء while-key event :

بعض البرامج تحتاج الى أداء اعمال بينما يضغط المستخدمون مفتاح الى اسفل ويستمرون فى الضغط. على سبيل المثال، يمكن ان يستخدم نظام تصميم رسوم جرافيكية اسلوب ضبط ظل لون من الفاتح الى الداكن بينما انت تضغط مفتاح سهم. قدر تطلق لعبة ما بينما انت مستمر فى ضغط قضيب المسافة. ان امكانيات هذه التقنية لحدود لها، وكما يوضح البرنامج التعليمى التالى، ان عملية البرمجة ليست صعبة.

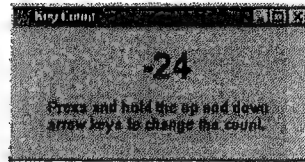
**على القرص المدمج:** ان واحداً من البرامج الخاصة بهذا الكتاب، وهو KeyCount، يوضح كيفية إنشاء while-key events. لقد تم إنشاء هذا البرنامج فى القائمة (٤-٢)، ويمكن ايضاً أن تجده على القرص المدمج المرفق بهذا الكتاب فى دليل الـ KeyCount. رغم ان الـ KeyCount ليس له قيمة عملية، الا انه يزيد ويقلل قيمة العدد الصحيح بينما انت تضغط وتستمر فى ضبط مفاتيح السهام لأعلى أو لأسفل - وبذلك يعرض الاسس الضرورية لإنشاء الـ while-key events.

قم بتشغيل الـ KeyCount. اضغط واستمر فى ضغط مفاتيح السهام لأعلى أو لأسفل لتزيد أو تنقص القيمة الصحيحة فى مركز نافذة البرنامج. اضغط



## الباب الرابع : برمجة لوحة المفاتيح والفأرة

Alt+F4 للإنتهاء يوضح الشكل (٤-٤) مظهر البرنامج. تعطى القائمة (٤-٢) Main.Pas unit الخاصة بالبرنامج.



شكل (٤-٤): عرض الـ KeyCount بعد ضغط مفتاح السهم المشير لأسفل لفترة

القائمة (٤-٢): KeycountMain.pas:

```
unit Main;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes,
  Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls;

type
  TMainForm = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Timer1: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key:
Word;
    Shift: TShiftState);
    procedure FormKeyUp(Sender: TObject; var Key:
Word;
    Shift: TShiftState);
    procedure Timer1Timer(Sender: TObject);
  private
```

```

{ Private declarations }
    Count: Integer;
    KeyPressed: Word;
public
{ Public declarations }
end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FormCreate(Sender: TObject);
begin
    Count := 0;
end;

procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;
    Shift: TShiftState);
begin
    if ((Key = vk_up) or (Key = vk_down)) then
    begin
        KeyPressed := Key;
        Timer1.Enabled := True;
    end;
end;

procedure TMainForm.FormKeyUp(Sender: TObject; var Key: Word;
    Shift: TShiftState);
begin
    Timer1.Enabled := False;

```

## الباب الرابع : برمجة لوحة المفاتيح والفأرة

end;

```
procedure TMainForm.Timer1Timer(Sender: TObject);
begin
  if KeyPressed = vk_up then
    Inc(Count)
  else if KeyPressed = vk_down then
    Dec(Count);
    Label1.Caption := IntToStr(Count);
end;

end.
```

يوضح البرنامج التعليمي التالي كيفية إنشاء تطبيق الـ KeyCount وكيفية برمجة while-key event. يوضح جدول (٣-٤) قيم الخصائص الهامة للبرنامج. جدول (٣-٤): خصائص الـ KeyCount

Component	Name	Property	Value
Form	MainForm	Caption	KeyCount
Label	Label1	Caption	0
		Font:Name	Arial
		Font:Size	24
		Font:Style	[fsBold]
		Caption	(enter text from
Label	Label2 Figure 4-4)	WordWrap	True
		AutoSize	False
		Interval	100
Timer	Timer1	Interval	100

لإنشاء التطبيق KeyCount:

١- لإنشاء النص أسفل نافذة البرنامج، قم بإضافة الـ Label (Label2) في جدول (٣-٤)، وقم بتغيير خاصية الـ WordWrap له لتكون True وخاصية الـ AutoSize لتصبح False. قم بتعديل حجم الـ object، ثم اختر خاصية الـ Caption. لا تضغط Enter وأنت تكتب. بدلاً من ذلك، اجعل السطور يحدث لها عملية wrap تلقائياً داخل حدود الـ component والنافذة.

٢- قم بإنشاء global variables، ولقد تم تسميتها بهذا الاسم لأنهما متاحان بشكل عام لكل أجزاء unit module. والمتغير الأول، ويسمى Count، من نوع ال Integer، يحمل القيمة المقابلة الحالية، المعروضة فى مركز النافذة. والمتغير الثانى، ويسمى KeyPressed، من نوع ال Word، يحمل قيمة ال virtual key. قم بتوضيح هذين المتغيرين بالتحويل الى نافذة ال Main.pas بالبرنامج، وأدخل هذه الأسطر بعد الكلمة الرئيسية var (استخدام القائمة ٤-٢ كإرشاد):

```
Count: Integer;
KeyPressed: Word;
```

٣- لكى تجعل العد يبدأ الصفر عندما يبدأ البرنامج التشغيل، اضغط مرتين قيمة ال OnActivate الخاص بال form وأدخل هذا التعيين بين البداية والنهاية:

```
Count := 0;
```

٤- يحتاج البرنامج إثنين من برامج ال event handlers الأخرى- لضغط المفتاح والآخر لإطلاق المفتاح. اضغط مرتين الأول لل OnKeyDown الخاص بال from وأدخل البرمجة من القائمة (٤-٢) بين البداية والنهاية. اقرأ هذه البرمجة بعناية. اذا كانت قيمة ال Key هى vk\_up أو vk\_down (الذان يمثلان مفاتيح الاسهم المشيرة لأعلى واسفل) إذن يحفظ البرنامج قيمة ال Key فى متغير ال KeyPressed العام. يربط البرنامج أيضاً بال Timer1 بالعبارة التالية. بعد هذه العبارة، يبدأ العداد باستدعاء ال OnTimer الخاص به:

```
Timer1.Enabled := True;
```

٥- قم بإنشاء event handlers آخر لل OnKeyUp الخاص بال from. وهذه ال event يوقف العداد بتحديد خاصية ال Enabled له ب False. بعد هذه العبارة، لا يستمر العداد ويستدعى ال OnTimer الخاص به:

```
Timer1.Enabled := False;
```

٦- وأخيراً، اختر ال Timer1 و قم بإنشاء ال OnTimer الخاص به (ال event الوحيد لل Timers). والبرمجة لهذا ال event تزيد أو تنقص ال Count اعتماداً على ما اذا كان ال KeyPressed يساوى vk\_up أو vk\_down. ويعرض ال event العداد البعد الحالى يتحويل ال Count الى string من خلال IntToStr وتعين النتيجة لل Caption الخاص ب Label1.

## الباب الرابع : برمجة لوحة المفاتيح والفأرة

ويمكن لمتغيرات الاعداد الصحيحة الاعداد الصحيحة ان تخزن قيم سائبة وموجبة من -٣٢٧٦٨ الى ٣٢٧٦٨ . ويمكن لمتغيرات ال Word ان تخزن القيم الموجبة فقط من صفر الى ٦٥٥٣٥ . والمتغيرات من نوعى البيانات والتي تمثل الاعداد الصحيحة فقط ، وليس الكسور .

عندما تحتاج الى إضافة أو طرح واحد من قيمة Ordinal ، استخدم كسور Dec وال Inc الخاصة ب Pascal كما يوضح ال Timer Timer . على سبيل المثال ، اذا كان ال Count متغير Integer ، فبدلاً من عبارات مثل :

Count := Count + 1; { ??? }

Count := Count - 1; { ??? }

يمكنك كتابة :

Inc(Count);

Dec(Count);

ولكن لا يمكنك استخدام ال Inc وال Dec على الخصائص ، وال procedure له تأثير إيجابى بسيط (ان وجد) على الاداء اثناء التشغيل . فهذه ليست استدعاءات ال procedure ، رغم انه هذا هو ما يبدو . إنهما فى الحقيقة قد تم تخصيصها لتعليمات ال CPU التى تزيد أو تنقص قيمة مخزنة فى مكان ما بالذاكرة . ويقوم ال Delphi optimizer الذكى بإحلال تعبيرات مثل  $x := x + 1$  و  $y := y - 1$  بتعليمات ال Inc() و Dec() المساوية .

### WinProcs، WinTypes، تغيير ال Enter الى Tab،

رغم ان ال Windows يتوقع ان يضغط المستخدمون مفتاح ال Tab للإنتقال من إحد حقول الإدخال الى حقل آخر ، كثير من مستخدمون الحاسب يضغطون Enter لهذا العرض . وللأسف ، ان ضغط ال Enter عادة ما يختار زر OK ، وهو الذى يغلق النافذة أو يحفظ المداخل الحالية قبل ان تتاح لك فرصة إتمامها جميعاً . وهذا يكون محبط للغاية .

وكحل بسيط لهذه المشكلة ، إتبع الخطوات التالية لإعادة برمجة لوحة المفاتيح بحيث ان ضغط ال Enter يعمل عملاً مشابهاً لضغط ال Tab :



١- اضعف ثلاث Edit component أو أكثر على ال form ان الNames الافتراضية ملائمة، ولكن يمكنك تغييرها اذا اردت.

٢- اضعف BitBtn وقم بتغيير ال Kind الخاص به ليصبح bkClose.

٣- حدد خاصية ال KeyPreview لل form ب True.

٤- قم بتغيير ال ActiveControl الخاص بال form ليصبح Edit1 (أو أى Name لأى Edit object).

٥- قم بإنشاء ال Enter event. OnKeyPress الخاص بال form. ادخل البرمجة الموضحة فى القائمة (٤-٣) اضغط F9 للتشغيل، ثم جرب ان تضغط Tab و Enter. تغلق النافذة عند اختيار زر ال Close. هذا النص موجود على القرص المدمج فى دليل ال KeyMouse، فى ملف Enter2Tab.pas.

القائمة (٤-٣)، جعل ال Enter يتصرف كال Tab

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  if Key = #13 then
  begin
    Key := #0;
    SelectNext(ActiveControl, True, True);
  end;
end;
```

تفحص عبارة ال if فى القائمة (٤-٣) اذا ما كان ال Key يمثل مفتاح ال Enter، الذى له قيمة ال ASCII العشرية. والتعبير #13 يعتبر طريقة اخرى لتمثيل قيمة ال ASCII الرمزية فى Pascal. اذا كان Key يساوى Enter، فإن عبارة تعيين سوف تحدد ال Key ب null. وهذا يبطل ال Enter.

و SelectNext تعتبر method من ال TWinControl class، وهى class اساسية لكل ال controls المرئية التى يمكن ان يكون لها input focus. استدع ال SelectNext لتقديم أو تأخير ال focus لل control التالى. ويتم تعريف ال method كالاتى:

```
procedure SelectNext(CurControl: TWinControl;  
GoForward, CheckTabStop: Boolean);
```

● **CurControl**، عادة، يجب ان يكون هذا مساوياً للـ **ActiveControl** الخاص بالـ **from**. رغم ذلك، قد تخصص **control object** بدلاً من **control** مختلف الى الـ **control** الحالي - على سبيل المثال، لاختيار حقل الـ **edit** الأول في الـ **from** لدخول البيانات.

● **GoForward**، حدد هذا الـ **parameter** بـ **True** لتنقل الـ **focus** للـ **control** التالي. حدده بـ **False** لتنقل الـ **focus** للـ **control** السابق.

● **CheckTabStop**، حدد هذا الـ **parameter** بـ **True** لنقل الـ **focus** للـ **control** التالي أو السابق في ترتيب الباب. حدده بـ **False** لنقل الـ **focus** للـ **control** التالي أو السابق بغض النظر عن اذا ما كانت خاصية الـ **TabStop** للـ **control** = **True** أم لا.

### إرسال الرسائل:

بدلاً من استدعاء الـ **SelectNext**، يمكنك أيضاً إرسال رسائل **Windows** لتغيير الـ **focus** الى **control** آخر على سبيل المثال في القائمة (٤-٣)، تستطيع ان تضع مكان استدعاء الـ **SelectNext** العبارة التالية، والتي تظهر كيف ترسل رسالة:

```
PostMessage(Handle, WM_NEXTDLGCTL, 0, 0);
```

ترسل العبارة رسالة الـ **Windows** وهي **WM\_NEXTDLGCTL** والتي تنقل الـ **focus** الى **control** التالي في الصف المرتب حسب **tab-order** تمثل نافذة الـ **form** - وهي قيمة يحتاجها الـ **procedure** للـ **Windows** المختلفة حتى تعرف أى من النوافذ يجب ان تتلقى رسالة معينة. وقيمتى الصفر لا يتم استخدامهما.

بالرغم من ان هذا الاسلوب يعمل جيداً، الا ان إرسال الرسائل بهذه الطريقة تشكل خطورة في ان تجعل الـ **code** غير قابلة للنقل وتعتبر اقل إختصاصاً بالـ **object** من استدعاء **SelectNext class method**. لا يوجد خطأ من الناحية الفنية في إرسال رسائل الـ **Windows** للـ **controls** والنوافذ الأخرى، ولكن عندما يوجد **method** يؤدي العملية، فمن الأفضل ان تستدعي هذا الـ **method**.

## Mouse Traps

بعض المستخدمين يتعاملون مع فأرة الحاسب وكأنهم فى سباق الانزلاق من على الحبال . وآخرون يتعاملون معها وكأنها تحتاج عجلات للسير . سواء احببتهم أم لا ، ان فئران الحاسب موجودة لتستقر ، ونجاح برمجياتك قد يعتمد على درجة إجابة برمجة الـ events الخادم بالفأرة الخاصة بتطبيقك ، كما سأوضح فيما بعد .

### الضغط مرة ومرتين:

لقد استخدمت بالفعل الـ events الخاصة بضغط الفأرة لأداء اعمال متنوعة . على سبيل المثال ، ان تقوم بعمل إذا ضغط المستخدم زر الفأرة الأيمن ، يمكنك ببساطة برمجة الـ `OnClick` لأي `component` أو غالبية الـ `components` . وللإستجابة للضغط مرتين ، استخدم الـ `OnDblClick` يمكنك برمجة أى من الـ `events` أو كليهما لنفس الـ `form` أو الـ `component` وتقع الـ `event` الخاصة بزر الفأرة الافتراضى ، وعادة هو الزر الأيسر . ولكن ، تذكر ان المستخدمين يمكنهم تغيير الزر الافتراضى بالـ `Windows Control Panel` .

ويعتبر الـ `OnClick` والـ `OnDblClick` ملائمتين لكثير ، ان لم يكن كل أعمال الفأرة ولكن للتوصل الى تحكم افضل فى الفأرة ، يمكنك إنشاء `create handlers` لهذه الـ `events` الثلاثة ، والتي هى متاحة للـ `forms` وغالبية الـ `components` .

● **OnMouseDown**: يستدعى عندما يضغط المستخدم لأي زر بالفأرة .

● **OnMouseMove**: يستدعى عندما يحرك المستخدم مؤشر الفأرة .

● **OnMouseUp**: يستدعى عندما يحرك المستخدم زر الفأرة .

وتتلقى الـ `Procedures` لهذه الـ `events` معلومات إضافية حول نشاط الفأرة- على سبيل المثال ، أى من الأزرار قام المستخدم بضغطه ، ومكان مؤشر الفأرة . ولفحص بعض هذه المعلومات ، ابدأ مشروعاً جديداً وقم بإنشاء الـ `event handler` خالية لكل من الـ `event` الثلاثة . إختبر تعريفات الـ `Procedure` . على سبيل المثال ، ها هو تعريف الـ `OnMouseDown` :

`procedure TForm1.FormMouseDown(`

```
Sender: TObject;  
Button: TMouseButton;  
Shift: TShiftState;  
X, Y: Integer);
```

● **Sender**: يمثل ال object الذى يتلقى هذا ال event. اذا أشرت أنان من components أو أكثر فى نفس برنامج ال event handler، يمكنك استخدام ال Sender لتحديد أى من ال object تم ضغطه. (ستعرف المزيد عن هذا فيما بعد).

● **Button**: واحد من القيم الثلاث: mbLeft، أو mbRight، أو mbMiddle. استخدم هذا ال parameter لتحديد أى من أزرار الفأرة قد قام المستخدم بضغطه.

● **Shift**: مجموعة من الصفر أو أكثر من القيم: ssCtrl، ssAlt، ssShift، ssLeft، ssRight، ssMiddle و ssDouble. استخدم هذا ال parameter لتحديد اذا ما كان المستخدم يضغط مفاتيح ال Shift أو ال Alt أو ال Ctrl (أو جمع بين هذه المفاتيح) وهو يضغط الفأرة. يمكنك أيضاً استخدام هذا ال parameter فى تحديد أى من أزرار الفأرة يضغطه المستخدم.

● **X, Y**: إحداثى النقطة لموقع ال form أو ال component التى يشير إليها مؤشر الفأرة، استخدم هذه القيم لإختيار ال objects فى النوافذ، لرسم الجرافيك، ولبدء عملية الضغط والسحب. (بعض النسخ من ال Delphi documentation تذكر خطأ أن هذه القيم مرتبطة بالشاشة. فى الواقع إنها متصلة بال client area فى النافذة، مع الأخذ فى الاعتبار أن الإحداثى 0,0 يمثل الركن الأعلى الى اليسار).

من الأفضل ألا تطلب استخدام زر الفأرة الأوسط، إلا فى الظروف الخاصة عندما تكون واثقاً من أن زر الفأرة الثالث متاحاً. إن أغلب الحاسبات الشخصية لها زران فأرة فقط.

يعرف ال OnMouseDown event handler نفس ال parameter مثل ال OnMouseUp، ولكنه يتم استدعاه عندما يترك المستخدم زر الفأرة. ويتلقى ال OnMouseMove event handler أيضاً نفس ال parameter إلا لل Button.

للتمييز بين زرى الفأرة الأيمن والأيسر، يجب أن تستخدم عادة ال Button parameter. على سبيل المثال، لكى تطلق صغيراً عندما يضغط المستخدم زر الفأرة الأيمن أو الأوسط - والذي قد تفعله لتشير الى أن البرنامج لا يميز بينهم - أضف ال event handler لل OnMouseDown event الخاص بال form وأدخل العبارة التالية :

```
if (Button = mbRight) or (Button = mbMiddle)
then MessageBeep(0);
```

**ملحوظة:** إن المصطلحات "أيمن" و "أيسر" يعتبران نسيين لتحديد مواصفات فأرة نظام المستخدم. تذكر، إن mbLeft يرجع الى زر الفأرة الأيسر، بينما يرجع mbRight الى الزر المقابل. إن المصطلحات "أيمن" و "أيسر" هما بقايا الأيام الخوالى لل Windows عندما لم يكن ممكناً تحويلهما.

Note

استخدم ال Shift parameter لتحديد ما اذا كان المستخدم يضغط مفتاحاً أيضاً أثناء ضغط الفأرة. على سبيل المثال، لتجعل ال speaker يصدر صغيراً اذا ضغط المستخدم زر الفأرة الأيسر أثناء ضغط مفتاح Ctrl، يمكنك استخدام هذه العبارة :

```
if (Button = mbLeft) and (ssCtrl in Shift)
then MessageBeep(0);
```

إن الجزء الأول من التعبير يعد صادقاً اذا كان ال button يساوى mbLeft، مشيراً الى ضغطه بالزر الأيسر، والجزء الثانى يختبر اذا ما كان ال ssCtrl أو أى قيمة ممكنة أخرى موجودة فى مجموعة قيم ال Shift التى مرت الى ال procedure.

يمكنك أيضاً استخدام ال Shift parameter لتحديد أى من الأزرار قام المستخدم بضغطه على سبيل المثال، تنتج العبارة التالية النتيجة المطابقة للعبارة السابقة، ولكن تفحص اذا ما كان ال ssLeft موجود فى مجموعة قيمة ال Shift :

```
if (ssLeft in Shift) and (ssCtrl in Shift)
then MessageBeep(0);
```

يمكنك تكملة هذا الأسلوب للإختبار عن مفاتيح أخرى مثل Alt :

```
if (ssLeft in Shift) and
```

```
(ssCtrl in Shift) and
(ssAlt in Shift)
then MessageBeep(0);
```

فى مثل هذه الحالات ، من الأسهل أن نحدد قيم المفاتيح الممكنة كمجموعة حرفية بين قوسين ، وقارنها بالـ Shift :

```
if [ssLeft, ssCtrl, ssAlt] = Shift
then MessageBeep(0);
```

إن كلا الأسلوبين ليسا متساويين بدقة ، على أية حال . إن الأسلوب الأول يجعل الـ speaker يصدر صفيراً إذا كانت الـ ssLeft ، و ssCtrl ، و ssAlt فى الـ Shift بغض النظر عما إذا كانت قيم أخرى موجودة فى نفس المجموعة . على سبيل المثال ، إن الأسلوب الأول سوف يمكن المستخدمين أن يستمروا فى ضغط مفتاح الـ Shift أيضاً ، ولكن الأسلوب الثانى لن يسمح بذلك . وكما يشير هذا ، إن المجموعات تعتبر فى متناول اليد ، ولكن يجب أن تفكر جيداً فيما تحاول الوصول إليه .

لا تخطئ بين الـ Shift parameter ومفتاح الـ Shift . لتحديد إذا ما كان مفتاح الـ Shift مضغوط أثناء الـ event للفأرة ، استخدم عبارة مثل :

```
if (ssShift in Shift) then
```

...

استخدم الـ x parameter و y لتحديد مكان مؤشر الفأرة أثناء إحداثيات التحريك ، أو الضغط ، أو الإطلاق للفأرة . على سبيل المثال ، لعرض أحداثيات فأرة عندما يضغط المستخدم زر الفأرة ، أضف اثنين من الـ Label component على الـ form وقم بإنشاء الـ OnMouseDown event handler الخاص بالـ form . (تأكد من إختيار الـ form event) أدخل العبارات التالية بين البداية والنهاية :

```
Label1.Caption := IntToStr(X);
Label2.Caption := IntToStr(Y);
```

قم بتشغيل البرنامج واضغط الفأرة داخل النافذة . توضح الـ labels موقع مؤشر الفأرة بالنسبة للـ client area النافذة .

## العودة الى ال sender:

لقد وعدت فيما سبق أن اشرح كيفية استخدام ال Sender parameter الذى يتم تمريره لكثير من ال event handlers مثل ال OnMouseDown . عندما يتشارك objects عديدة فى event handler ، يمكنك استخدام ال Sender لتحديد أى من ال object قد تلقى ال event .

وكمثال على هذا الأسلوب المفيد، إتبع هذه الخطوات :

١- فى نافذة ال form ، أدخل ثلاثة أزرار يحملون الاسماء الافتراضية وهى Button1 ، Button2 ، و Button3 . (فكرة: أضف زرأ واحداً، اضغط Ctrl+C لتكوين نسخة منه فى الحافظة . ثم اضغط Ctrl+V مرتين لإنشاء نسختين أخرتين . هذه الطريقة أسرع من إضافة objects متعددة واحداً تلو الآخر ، ولكن عليك أيضاً تعديل ال Captions الخاصة بالأزرار ، وإلا سوف يحملون جميعاً المسمى Button1).

٢- اضغط واسحب ال rubber band outline لإختيار الأزرار الثلاثة . سوف ترى الأزرار قد تم إختيارهم وتظهر نافذة ال Object Inspector الآن ال events والخصائص المشتركة بين الأزرار .

٣- اختر ال Events page بال Object Inspector ، واضغط مرتين قيمة ال OnMouseDown event . هذا ينشئ ال event handler التى يتشارك فيه الأزرار الثلاثة . بعبارة أخرى ، إن ضغط أى زر أثناء وقت التشغيل يستدعى نفس ال procedure .

٤- من الواضح أن ال procedure المتشارك فيه يحتاج أسلوباً لتحديد أى من الأزرار قمت بضغطه ، هنا يأتى دور ال Sender . فى ال OnMouseDown procedure ، أضف العبارات وتعريف المتغيرات الموجودة فى القائمة (٤-٤) . على القرص المدمج ، يوجد هذا النص فى دليل ال KeyMouse فى ملف ال WhichButton.pas .

٥- اضغط F9 لتشغيل البرنامج . عندما تضغط زرأ ، تؤكد رسالة أى من ال button object قمت بإختياره .

القائمة (٤-٤)، مثال ال Sender

```

procedure TForm1.Button1MouseDown(Sender: TObject;
  Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  s : string;
begin
  if Sender = Button1 then
    s := 'Button1'
  else if Sender = Button2 then
    s := 'Button2'
  else if Sender = Button3 then
    s := 'Button3'
  else s := 'Unknown Object';
  ShowMessage('You clicked ' + s);
end;

```

في القائمة (٤-٤)، تقوم عبارة ال if-then-else بمقارنة ال Sender بـ Button1، Button2، و Button3. وإعتماداً على قيمة ال Sender، يحدد البرنامج ال string variable لاسم الزر المناسب. وعبارة ال sets الأخيرة تحدد المتغير للـ Unknown Object فقط في حالة أن يكون ال Sender لا يتناسب مع واحداً من الأزرار الثلاثة.

والعبارة الأخيرة في القائمة (٤-٤) تستدعي ال ShowMessage لعرض ملحوظة حول الزر الذي قمت بإختيار. يمكنك تمرير أى ShowMessage. لاحظ كيف أن علامة زائد تضاف للـ string المعين بواسطة عبارة ال if-then-else السابقة للـ string الحرفي "You clicked".

وعبارة ال if-then-else في القائمة (٤-٤) تعتبر عبارة واحدة، وليست عدة عبارات. ولذلك فهي منفصلة عن العبارة التالية لها فصلة منقوطة.

عند استخدام ال Sender، فإنك غالباً ما تحتاج أن تخبر ال Delphi أى نوع من ال object يمثل ال parameter. يمكنك عندئذ تعيين قيم لخصائص ال



Sender . على سبيل المثال، لتغيير الـ Caption لكل زر بعد أن يتم إختياره، فإن العبارة التالية لا تنفع :

```
Sender.Caption := 'Clicked'; // ???
```

لا يمكنك أن تفعل هذا لأن الـ Sender من نوع الـ TObject ، الذى ينحدر منه كل Delphi component . بمعنى آخر، إن كل الـ component تعتبر TObject . للتوصل الى خصائص الـ Caption بالأزرار، باستخدام عبارة with-do لتخبر Delphi إن الـ Sender هو فى الواقع object TButton - أضف العبارة التالية فى الـ procedure الموجود فى القائمة (٤-٤) :

```
with Sender as TButton do
```

```
Caption := 'Clicked';
```

إن تغيير الـ with-do يخبر الـ Delphi أن يتعامل مع الـ Sender كأنه TButton object ، والذى له خاصية Caption . عندما تقوم بتشغيل البرنامج الذى، تم تعديله، تتغير كل مسمى للزر الآن لتصبح Clicked بعد أن تختارها . إن استخدام "as" لا يحول الـ Sender الى TButton object - إنه فقط يرى الـ Sender كأنه ذلك النوع الخاص من الـ object . إنها مسئوليتك أن تضمن أن الـ Sender حقيقة هو نوع الـ object الذى حددته . لا يمكنك استخدام الكلمة الرئيسية as لتحويل الـ object الى آخر من نوع مختلف .

### مؤشرات الفأرة:

يمكنك تحديد واحداً من الـ objects المتعددة لمخزون المؤشرات لـ أى form أو component . على سبيل المثال، ابدأ تطبيقاً جديداً، وقم بتغيير خاصية الـ Cursor للـ from من الـ crDefault الى الـ crCross . اضغط F9 للتشغيل، وحرك مؤشر الفأرة فوق نافذة برنامج وبعيداً عنها . يتغير المؤشر تلقائياً الى شكل علامة الزائد، ثم يعود الى شكل السهم عندما تحركه بعيداً عن الـ client area بالنافذة .

قد يكون للـ Components أيضاً شكل مؤشرات مرتبطة بها والتى تشير الى أنواع العمليات التى يمكن للمستخدم أن يختارها أو يؤديها . على سبيل المثال، أضف الـ GroupBox component الى الـ form وحدد خاصية الـ Cursor لها بـ crHSplit . اضغط F9 والتشغيل . عندما تحرك مؤشر الفأرة داخل الـ GroupBox

## الباب الرابع: برمجة لوحة المفاتيح والفأرة

يتحول شكل علامة الزائد الى شكل split-cursor، والذي يظهر عند حدوث عملية الـ clicking أو حدوث الـ dragging لتقوم الفأرة بتغيير الأحجام للـ split window.

### :Custom mouse cursors

يمكنك أيضاً إنشاء أشكال المؤشر الخاصة بك لأي form أو component وهذا الأمر ليس صعباً، ولكن يتطلب بعض العمل الزائد وقليل من البرمجة.

١- قم بتشغيل الـ Image Editor الخاص بـ Delphi من قائمة الـ Tools لإنشاء أو تعديل صورة المؤشر، وهى عادة لـ 32x32 pixels bitmap ذات لونين.

٢- احفظ صورتك فى ملف له إمتداد .cur، يمكنك أيضاً أن تجد العديد من ملفات المؤشر على bulletin boards أو الـ public domain و shareware. وكذلك عليك أن تفحص الملفات فى دليل الـ Images\Cursors الخاص بـ Delphi.

على القرص المدمج: اذا لم تكن قد قمت بتركيب ملفات المؤشر الخاصة بـ Delphi وليس لديك وقتاً لإنشاء أخرى، استخدم ملف الـ Sample.cur فى الدليل الفرعى Data على القرص المدمج المرفق بهذا الكتاب.



٣- يجب عليك بعد ذلك أن تقوم بإدخال المؤشر كـ Resource فى تطبيق Delphi. يتم تخزين الـ Resource داخل ملف الـ .exe الخاص بالبرنامج، ولكن أثناء التطوير، من الأفضل أن تقوم بتخزينهم منفصلين فى ملفات .res. إم عملية الـ Compiling والـ linking للبرنامج يربط ملف بيانات الـ .res فى ملف الـ code- لا يجب عليك توزيع ملفات الـ .res للمستخدمين النهائيين.

٤- استخدم الـ Image Editor الخاص بـ Delphi بإنشاء ملف resource، من الممكن أن يسمى Cursor.res. استخدم File/New لإنشاء ملف resource جديد، ثم اضغط يميناً داخل نافذة resource لإنشاء bitmap، مؤشر، وأيقونة تكون resource objects. وهذه الـ objects تأخذ الاسماء الافتراضية مثل CURSOR\_1. يمكنك تغيير هذه الاسماء اذا أردت.

احفظ ملف resource، وانسخه في دليل مؤقت. على القرص المدمج، قد تستخدم ملف الـ Cursor.res في دليل الـ Data.

٥- يمكنك استخدام الـ Image Editor لإنشاء ملف cursor image، وكذلك أنواع أخرى من الصور مثل bitmaps. ولكن كـ resource، يكون للمؤشر اسم resource والذي يمكن للبرنامج أن يشير إليه- صور المؤشر الباهتة لا تحمل اسماء cursor resource كذلك مرتبط مباشرة داخل الـ exe. الخاص بالبرنامج.

لاستخدام cursor resource، يجب أن يقوم البرنامج بهذه الخطوات الثلاث:

- تجميع الـ resource في ملف الـ code الخاص بالبرنامج.
  - تحميل الـ resource في الذاكرة أثناء وقت التشغيل.
  - حفظ الـ handle الخاصة بالـ resource للاستخدام في البرنامج.
- يهتم Delphi بالخطوة الأولى عندما تقوم بالـ compile والـ link للبرنامج. إنك تؤدي الخطوتين الأخرتين بإضافة تعليمات وعبارات للبرنامج.

٦- ابدأ تطبيقاً جديداً، وفي ملف unit source code، أدخل هذا الأمر بعد الكلمة الرئيسية implementation، بإتباع الأمر المشابه الموجود بالفعل والذي يقرأ ملف dfm. الخاص بالبرنامج. ويجب أن يبدو السطران كما يلي:

```
{ $R *.DFM }
```

```
{ $R Cursor.res }
```

٧- قم بتغيير اسم الملف Cursor.res، إذا لزم الأمر. (يمكنك أيضاً أن تدخل Cursor بدون إمتداد اسم الملف res). ويخبر الأمر الـ linker بأن يقرأ ملف الـ resource، والذي قد يحتوى على resource متعددة، ويضاف الـ resources في ملف الـ exe. يمكنك إضافة أى resources من ملف res. باستخدام نفس نوع الأمر، ويمكنك إدخال أى عدد من الأوامر كما تحتاج في ملف Pascal unit. ولكن في برمجة Delphi، إنك تقوم بهذا فقط للـ bitmaps، والمؤشرات في الأيقونات كما هو موضح هنا. وأدوات Delphi الأخرى مثل Menu editor تجعل استخدام الـ resources أمر غير ضروري.

**ملحوظة:** إن أمر (اسم ملف \$R) يتم تنفيذه أثناء عملية linking و compilation ، ولكن ليس في وقت التشغيل . لا يجب عليك أن توفر ملف الـ res. مع التطبيق التام.

Note

٨- بعد ذلك ، أضف الـ code لبرنامج لتحميل صورة المؤشر من ملف exe. الى الذاكرة في وقت التشغيل . على سبيل المثال ، لتحميل وتشغيل المؤشر قبل أن تصبح نافذة الـ form مرئية ، قم بإنشاء handler للـ OnCreate event الخاص بالـ form . قم بتعريف ثابت كعدد صحيح لتعريف المؤشر . على سبيل المثال ، قم بإدخال هذه الأسطر قبل كلمة البدء الرئيسية :

```
const
    newCursorID = 2;
```

يمكنك استخدام أى قيمة عدد صحيح موجبة لم يتم استخدامها من قبل لمؤشر آخر . اذا كان لديك مؤشرات متعددة . (أو resource أخرى) ، قم بتعريف ثوابت أعداد صحيحة لكل منها .

٩- قم بإنهاء OnCreate event handler بإضافة برمجة لتحميل الـ cursor resource فى الذاكرة . توضح القائمة (٤-٥) procedure الأخير .

#### القائمة (٤-٥)، تحميل cursor resource

```
procedure TForm1.FormCreate(Sender: TObject);
const
    newCursorID = 2;
begin
    Screen.Cursors[newCursorID] :=
        LoadCursor(HInstance, 'Cursor1');
    Cursor := newCursorID;
end;
```

توضح العبارتان فى القائمة (٤-٥) الطريقة الصحيحة لتحميل cursor resource واستخدامه فى الـ form أو component آخر . ويمثل متغير Screen ، وهو من نوع الـ TScreen ، يمثل مختلف أحجام الـ display ، انظر الـ TScreen فى

online help الخاصة بـ Delphi للحصول على مزيد من المعلومات حول هذا النوع (الزائد من البيانات). ويقدم الـ Screen.Cursors قائمة بصور المؤشرات، ومن الناحية الفنية يسمى array property . لتعيين قيمة للـ Cursors ، استخدم cursor images function Windows LoadCursor كما هو موضح هنا لتحميل cursor داخل الذاكرة، والذي يسمى هنا 'Cursor1'. (وهذا الـ string لا يعتبر حالة حساسة يمكنك إدخال اسماء resource بحروف صغيرة أو حروف كبيرة). الـ HInstance argument يشير الى الوظيفة الحالية أو instance الخاص للبرنامج. ويستخدم Windows الـ HInstance . لإيجاد ملف الـ .exe المناسب الذى يحتوى على الـ cursor resource المحدد.

قم بتعيين نتيجة الـ LoadCursor الخاصية الـ Cursors array ، وحدد معرف الثابت الواقع بين القوسين . لتحديد المؤشر للـ form ، قم بتعيين ثابت الـ Cursor للـ form أو لـ component آخر .

**ملحوظة:** يحدد الـ Cursor الفردى المؤشر الذى يتم تعيينه حالياً، و plural Cursors تشير الى array property التى تعطى كل المؤشرات المتاحة. عندما تقوم بتشغيل البرنامج، قم بتحريك مؤشر الفأرة داخل النافذة لترى شكل المؤشر الجديد.



يمكنك الاستفسار عن صلاحية التعيين للـ Cursors فى القائمة (٤-٥). بالرغم من المظاهر، ان الـ Cursors لا يعتبر Pascal array ، إنما array property . وكذلك، ان الـ Cursors تستدعى فى الحقيقة write procedure التى تعود وتضيف cursor resources . ان Cursors[X] يبدأ البحث عن مؤشر معرف على أنه X . والتعيين للـ Cursors[X] يحذف أى مؤشر معرف على أنه X قبل إدخال مؤشر جديد فى الخاصية . ومن وراء الستار، يتم التعامل مع الـ Cursors على أنه linked list وهو من الناحية الفنية sparse associated array ، ولكن هذه الحقيقة يخفيها الـ TScreen ببراعة.

**فكرة:** لا يستطيع Delphi تحميل ملفات .res ذات ١٦ بت - على سبيل المثال، تلك التى هى من تطبيقات الـ Windows 3.1 . لتحديثها الى ٣٢ بت، قم بإنشاء ملفات resources جديد باستخدام الـ Image Editor .

Tip



## الرسم بالفأرة:

على القرص المدمج: باستخدام تقنيات الفأرة التي عرفت في هذا الباب، يمكنك برمجة عمليات الضغط والسحب لأغراض متعددة. ان التطبيق التالي، Sketch، يوضح الاسس الضرورية للضغط والسحب. ضع هذا المشروع في دليل ال Sketch. (يوجد برنامج مشابه في ادلة ال Delphi-أستخدم المتوفر في القرص المدمج المرفق بالكتاب). قم بتشغيل Sketch واضغط واسحب الفأرة داخل نافذة التطبيق لرسم اشكال بسيطة. اضغط مرتين مؤشر الفأرة داخل النافذة هو ال display توضح القائمة (٤-٦) source code للبرنامج.



القائمة (٤-٦): Sketch\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,  
  Controls, Forms, Dialogs, Menus;
```

```
type
```

```
  TMainForm = class(TForm)
```

```
    MainMenu1: TMainMenu;
```

```
    Demo1: TMenuItem;
```

```
    Erase1: TMenuItem;
```

```
    Exit1: TMenuItem;
```

```
    N1: TMenuItem;
```

```
    procedure FormMouseDown(Sender: TObject;  
      Button:
```

```
      TMouseButton; Shift: TShiftState; X, Y: Integer);
```

```
    procedure FormMouseMove(Sender: TObject;  
      Shift:
```

```
      TShiftState; X, Y: Integer);
```

```
    procedure FormMouseUp(Sender: TObject; Button:
```

```

TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure FormDb1Click(Sender: TObject);
procedure Erase1Click(Sender: TObject);
procedure Exit1Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
  Dragging: Boolean;
end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FormCreate(Sender: TObject);
begin
  Dragging := False;
end;

procedure TMainForm.FormMouseDown(Sender: TObject; Button:
  TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Dragging := True;
  Canvas.MoveTo(X, Y);
end;

```

## الباب الرابع : برمجة لوحة المفاتيح والفأرة

```
procedure TMainForm.FormMouseMove(Sender: TObject; Shift:
  TShiftState; X, Y: Integer);
begin
  if Dragging then
    Canvas.LineTo(X, Y);
end;
```

```
procedure TMainForm.FormMouseUp(Sender: TObject; Button:
  TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Dragging := False;
end;
```

```
procedure TMainForm.FormDbClick(Sender: TObject);
begin
  Erase1Click(Sender);
end;
```

```
procedure TMainForm.Erase1Click(Sender: TObject);
begin
  Canvas.Brush := Brush; { Assign form's brush to Canvas }
  Canvas.FillRect(MainForm.ClientRect); { Repaint
  form background }
end;
```

```
procedure TMainForm.Exit1Click(Sender: TObject);
begin
  Close;
end;
```

end.

اتبع هذه الخطوات لإعادة إنشاء تطبيق ال Sketch ولتعرف كيف تنفيذ عمليات الضغط والسحب للفأرة:



١- اذهب الى نافذة ال unit، وفي TMainForm class، عرف المتغير Dragging، من نوع ال Boolean. ([انظر القائمة (٤-٦)] يحدد البرنامج هذا المتغير بـ False ليشير الى ان عملية الضغط والسحب قد بدأت. عندما يصبح False، يشير ال Dragging بذلك الى عمليات الفأرة الطبيعية.

٢- حدد المتغير ال Dragging بـ False بإضافة ال OnCreate event الخاص بال form. اضع هذه العبارة الى ال procedure:

Dragging := False;

٣- لتنفيذ عملية الضغط والسحب، اضع ال procedure لكل من ال form events الثلاث، OnMouseDown، OnMouseMove، و OnMouseUp. أضف البرمجة بعد الخطوة الأخيرة من هذا البرنامج التعليمي.

٤- أضف ال menu component، وادخل اوامر لمسح ال display وانهاء التطبيق. يمكن ان يستخدم أمر المسح عبارات مثل ال client area فى النافذة بلون الخلفية الحالى. أولاً، قم بتعيين خاصية ال Brush لل form التابعة ال Canvas's Brush، ثم استدع ال FillRect للملى ال Canvas بذلك اللون (يشرح الباب ١٣ المزيد حول ال Canvases والرسوم الجرافيكية):

Canvas.Brush := Brush;

Canvas.FillRect(MainForm.ClientRect);

٥- خصص ال Erase procedure الخاص بك ال OnDbClick التابع لل form. يمكنك الآن إختيار أمر القائمة أو الضغط مرتين على الفأرة لمسح محتويات النافذة.

٦- اضغط F9 لإجراء عمليتى ال Compile وال link وتشغيل التطبيق.

عندما تضغط الفأرة، يحدد OnMouseDown ال Dragging flag بـ True، ويستدعى ال MoveTo procedure ليماثل موضع الرسوم الجرافيكية لإحداثى الفأرة الحالى.

وتأخذ أية رسوم مرئية مكاناً فى هذا الموضع. عندما تطلق الفأرة، يحدد ال OnMouseUp ال Dragging بـ False، مما يلغى عملية الضغط والسحب.

## الباب الرابع : برمجة لوحة المفاتيح والفأرة

عندما تحرك الفأرة، يقوم الـ `OnMouseMove` بفحص اذا ما كان الـ `Dragging` محدد بـ `True` - اذا كان كذلك، فإنه يستدعي الـ `Canvas's LineTo` procedure ليرسم خطاً مرئياً من احدث موقع جرافيك حالي الى وضع الفأرة الجديد، والذي يصبح الموقع الجرافيكي لعملية الرسم التالية. اذا كان الـ `Dragging` محدد بـ `False`، لا يقوم الـ `OnMouseMove` بأداء اية اعمال.

ويعرض تطبيق الـ `Sketch` الضغط والسحب فقط فهو لا يقوم بالحفاظ على رسومك. بالإضافة الى ذلك، ان تغطية الـ `Sketch display` بأية نافذة أخرى يؤدي الى مسح الرسوم. يشرح الباب الثالث عشر اساليب إنشاء مزيداً من التطبيقات الجرافيكية العملية.

### افكار للاستخدام الخبير

• ان الـ `FormKeyDown` procedure التابع للـ `KeyCount` يعين قيمة المفتاح الحالي للـ `KeyPressed` قبل تشغيل العداد. ان عكس ترتيب هاتين العبارتين سيكون خطأ لان الـ `timers` تبدأ في العمل بمجرد ان تحدد خاصية الـ `Enabled` لهم بـ `True`. ولا تفعل هذا حتى بعد تحضير أى وكل الظروف المطلوبة من جانب الـ `OnTimer event` الخاص بـ `Timer component`.

• يمكنك استخدام الـ `resource editor` مثل الـ `Resource Workshop` الخاص بـ `Borland C++` أو `C++ Builder`، لإنشاء ملفات `.res` للاستخدام في تطبيقات `Delphi`. قم بضم ملف الـ `resource` في ملف الـ `code` بإضافة الامر `{ $R اسم ملف .res }` يتبع كلمة الـ `implementation` الرئيسية الخاصة بالـ `unit`.

• ان واضعي البرامج الخبراء يستخدمون الحروف الكبيرة والصغيرة للمساعدة على التمييز بين تعريفات البرنامج. وطبقاً للعرف يبدأ تعريف الثوابت مثل `newCursorId` وبأحرف صغيرة، والمتغيرات مثل الـ `Count` تبدأ بأحرف كبيرة. وتبدأ الـ `Procedures` مثل `LoadCursor` ايضاً بأحرف كبيرة وهذه ليست قواعد صارمة، ويمكنك اتباع أسلوب آخر اذا اردت. فقط تأكد من اتباع بعض الأعراف لتجعل الـ `code` مفهوم نوعاً ما.

• ان برمجة مفتاح ال Tab تعتبر صعبة في الغالب بسبب الطريقة التي يستخدم ال Windows بها هذا المفتاح لتحويل ال focus من control الى آخر. للتعرف على Tab-key events في تطبيقات Delphi، اُضف البرنامج في OnKeyUp event . (ان ال OnKeyDown وال OnKeyPress لا يتلقيان الضغط على مفتاح ال Tab). على سبيل المثال، ادخل البرمجة من القائمة (٤-٧) في برامج OnKeyUp event handler الخاص بال form لعرض رسالة عندما تضغط ال Tab أو Shift+Tab. وهذا النص موجود على القرص المدمج في دليل ال KeyMouse في ملف Tab.pas.

القائمة (٤-٧): تلقى الضغط للـ Tab أو Shift+Tab

في OnKeyUp event handler الخاص بال form

```
procedure TForm1.FormKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
var
  S: string;
begin
  if Key = vk_Tab then
    begin
      S := 'Tab Key Pressed';
      if ssShift in Shift then
        S := 'Shift-' + S;
      ShowMessage(S);
    end;
end;
```

### المشروعات التي يمكنك تجربتها

٤-١: قم بإنشاء نسخة لوحة مفاتيح من برنامج ال Sketch. إنك تحتاج قليل من قيم ال Integer لتمثيل موضع الرسم الحالي. قم ببرمجة ال event handlers الموجهة للزيادة والنقصان في الوضع الحالي عندما يضغط المستخدم مفاتيح السهام بلوحة المفاتيح، واستدع

CanvasMoveTo وال LineTo methods التابعين لل form

لرسم خطوط .

٢-٤ : اكتب برنامج اختياري يعرض رسالة أو صفيراً لل speaker (الاستجابة بالضبط لا يهم نوعها) عندما يتم ضغط مفتاحي الفأرة . إنك تحتاج الى Boolean flag الذي ، عندما يكون True ، يشير الى ان زر الفأرة الأول قد تم ضغطه ولكنه لم يترك ال event handler للزر الثاني يمكن ان يفحص ما اذا كان ال flag بـ True ، وبهذه الطريقة ، يبرمج الزران على ان يعملوا بطريقة مشابهة لمفاتيح ال Ctrl وال Alt .

٣-٤ : اضع custom pen-shaped cursor الى التطبيق ال Sketch .

٤-٤ : مستوى متقدم . اضع اوامر قائمة للاختيار فيما بين مؤشرات مختلفة لتطبيق ال Sketch . ملحوظة : قم بتحميل كل cursor resource ، تم تعريفه بثابت فريد ، وفي ال procedures لاوامر القائمة ، قم بتعيين الثابت الذي تم اختياره لخاصية ال cursor التابعة لل form .

### ملخص:

• اضع keyboard handling للتطبيقات بطريقتين : استخدم ال Edit أو أى edit component أخرى ، أو اضع ال procedure ال OnKeyDown وال OnKeyPress وال OnKeyUp الخاصة بال form .

• يمثل ال OnKeyDown وال OnKeyUp مفاتيحاً تستخدم ال virtual key codes مثل ال vk\_up ال vk\_shift . وتمثل ال OnKeyPress مفاتيح مثل رموز ال ASCII .

• يستخدم ال Windows ال input focus لتحديد اين يرسل ال events بلوحة المفاتيح . بشكل طبيعي ، تذهب مدخلات لوحة المفاتيح الى ال component الذي يتم اختياره حالياً . حدد خاصية ال KeyPreview لل form بـ True لتعيد توجيه أى نشاط للوحة المفاتيح موجه الى ال component object الحالي . المحدد لأستقبال أول event من لوحة المفاتيح .

- استخدم الـ Timer component لإنشاء الـ while-key، والذي يفيد في أداء الأعمال تكراراً بينما يستمر المستخدم في ضغط مفتاح بعينه.
- استخدم الـ OnKeyPress event لتعديل قيم المفتاح. على سبيل المثال، استخدم التقنية الموجودة في هذا الباب لتغيير مفاتيح الـ Tab لتصبح Enter بحيث يمكن للمستخدمين أن يضغطوا Enter لتحريك input focus من input control إلى آخر.
- إن event الفأرة OnClick و OnDbClick تعد كافية لغالبية الأعمال التي تقوم بها الفأرة. لمزيد من التحكم في نشاط الفأرة، أضف الـ OnMouseDown procedure، والـ OnMouseMove، والـ OnMouseUp. وتتلقي هذه الـ events معلومات إضافية مثل ما إذا كان مفتاحاً مضغوطاً وموقع مؤشر الفأرة.
- استخدم الـ Sender parameter، والذي يتم تمريره لأغلب الـ event handlers، لتحديد أى من الـ objects قد تتلقى الـ events. وهذا الأسلوب غاية في الإفادة عندما يتشارك عدة objects في نفس الـ event handler.
- قم بتغيير خاصية الـ Cursor لـ component أو form لعرض شكل cursor مختلف عندما تتحرك الفأرة فوق الـ form أو الـ component يمكنك الاختيار من بين عدد من المؤشرات المتوفرة.
- يمكنك أيضاً تصميم وتحميل مؤشر محدد كـ resource في ملف الـ .res. استخدم أمر {اسم ملف \$R} لضم بيانات ملف الـ resource إلى ملف الـ .exe. و event. مثل الـ OnCreate، استدع Windows LoadCursor function وقم بتعيين النتيجة للـ Cursors array الخاصة بالـ component وبالـ form باستخدام ثوابت الأعداد الصحيحة. قم بتعيين الـ index لخاصية الـ Cursor (singular) لتشغيل صورة المؤشر المخصص.
- في الباب القادم، سوف نشرح واحدة من أهم خصائص واجهة التطبيق للمستخدم في غالبية تطبيقات الـ Windows: وهو الـ menu bar تعتبر برمجة الـ menu bar أمر بسيط مع الـ Menu Designer الخاص بـ Delphi ويمكنك استخدام عدد من خيارات قوائم برنامجك، مثل الـ check marks، الأيقونات والقوائم الديناميكية.

# الباب الخامس بناء القوائم

## محتويات الباب:

• Components

• Pop-up menus

• Floating pop-up menus

• Dynamic menus

• Form frameworks

• Data modules

• خطوات نحو Menu designer

• ActionLists

كما يعرف كل مستخدم للـ Windows ، ان الـ pop-up menu ما هي الا كشف مفصل للأوامر ، مثل Open و Exit . في أغلب التطبيقات ، يعطى الـ menu bar اسم واحدة أو أكثر من pop-up menus . يمكنك ايضاً إنشاء floating pop-up menu تظهر عندما تضغط زر الفأرة الأيمن .

ان تصفح قائمة مصممة بعناية يشبه ارتداء معطفك المفضل والانزلاق داخله ، ولن تبذل جهداً كبيراً حتى تجعل قوائم تطبيقك مريحة في الاستخدام كما يوضح هذا الباب ، Delphi يقدم ثلاث menu components وعدد من التقنيات المتعلقة بها لبناء قوائم سهلة الاستخدام ذات خصائص معقدة سوف يقدرها المستخدمون .

## Components

ان ال menu component الثلاثة الخاصة بـ Delphi هي :

● **MainMenu**: أستخدم هذا ال component لإنشاء ال menu bar للنافذة، والذي يتم عرضه دائماً تحت ال top border و ال caption . لإنشاء قوائم ديناميكية وهى القوائم التى تتغير اثناء وقت التشغيل طبقاً لعمليات البرنامج المتنوعة، مثل فتح نافذة جديدة- يمكنك إضافة MainMenu متعددة على forms وتتبع التعليمات الموجودة فى هذا الباب لدمج اوامر القوائم .

● **PopupMenu**: أستخدم هذا ال component لإنشاء floating pop-up menu، التى تظهر عندما يضغط المستخدم زر الفأرة الأيمن فوق ال client area فى النافذة . يمكنك أيضاً تعريف طرق اخرى لعرض floating pop-up menu فى أى موضع من الشاشة .

● **MenuItem**: كل بند فى ال pop-up menu أو floating pop-up menu يعد object من ال TMenuItem class ولكن MenuItem component لا يظهر على VCL palette بالرغم من انه من الاسهل إنشاء هذه ال objects بال Menu Designer الخاص بـ Delphi، كما يوضح هذا الباب، يمكنك أيضاً استخدام عبارات البرنامج أو resource scripts لإنشاء menu items . (بالرغم من ان ال TMenuItem هو اسم ال class، الا اننى اشير الى هذا ال component بـ MenuItem من الآن فصاعداً لكى يكون متماشياً مع اسماء ال components الأخرى).

## Pop-Up Menus

من الناحية الفنية، تعتبر ال pop-up menu نافذة، والتى تراها عندما تفتح بنداً على ال menu bar للنافذة . يمكن ان تظهر floating pop-up menu فى أى مكان على الشاشة . بخلاف . هذا يعتبر نوعان مثل بعضهما . توضح هذا الفصل تقنيات البرمجة التى تنطبق على كلا النوعين من ال pop-up menu أستخدم MainMenu component لإنشاء pop-up menu على ال menu bar للنافذة . أستخدم ال PopupMenu component لإنشاء floating pop-up menu .

## :Main menus

ان معظم برامج ال Windows لها قائمة رئيسية ، والتي يمكنك إنشاءها باستخدام ال MainMenu . وكما عرفت من النماذج الأخرى بهذا الكتاب ، فإنك تنشئ قائمة رئيسية بادخال ال MainMenu على ال form الرئيسية للبرنامج . اضغط ال object مرتين لفتح ال Menu Designer الخاص بـ Delphi ، وادخل اوامر برنامجك .

قم بإنشاء قائمة رئيسية للبرنامج باتباع الخطوات الرئيسية الثلاث التالية :

١- استخدم ال ProjectOptions لاختيار Forms page tab وحذف ال Main form لل form الرئيسية للبرنامج . هذا هو التحديد الافتراضى للتطبيقات ذات النافذة الواحدة .

٢- أضف ال MainMenu ، على ال form الرئيسية .

٣- حدد خاصية ال Menu لل form باسم ال MainMenu object's .

بالرغم من ان قائمة رئيسية واحدة فقط يمكن ان تكون فى تطبيق ما ، الا من خلال شرطين يمكن لأى form ، حتى ال dialog box ، ان يكون لها menu bar . ببساطة أضف ال MainMenu object على ال form . والشرطان هما : خاصية ال BorderStyle لل form قد لا تساوى bsDialog ، وال FormStyle لها قد لا تساوى MDI child . (قد تمتلك نوافذ ال fsMDIChild ال MainMenu object ، ولكن كما يوضح الباب الخامس عشر ، تطوير تطبيقات ال MDI ، هذه القوائم قد تم دمجها فى menu bar رئيسية .

## : Floating pop-up menus

لإنشاء floating pop-up menu والتي تظهر عندما يضغط المستخدمون زر الفأرة الأيمن ، اتبع هذه الخطوات (وكذلك انظر فصل " floating pop-up menu " مؤخراً فى هذا الباب) :

١- أضف ال PopupMenu1 object على ال form . يجعل Delphi اسم ال PopupMenu1 object ، حسب النظام الافتراضى .



٢- اضغط على PopupMenu1 object مرتين، واستخدم ال Menu Designer لإدخال أوامر القائمة.

٣- عين ال Popup Menu1 الخاصية ال PopupMen التابعة لل form :

### : Menu items

يعتبر كل عنصر فى قائمة ال object من TMenuItem class. يقوم ال Delphi تلقائياً بإنشاء هذه ال objects عندما تصمم قوائمك مع ال Menu Designer. يمكنك رغم ذلك، إنشاء MenuItems مع عبارات البرنامج، وغالباً ما تشير الى هذه ال objects بالعبارات. على سبيل المثال، لإضافة علامة صح الى امر القائمة، قم بتعيين True الخاصية ال Checked الخاصة ال MenuItem، يمكنك ايضاً أداء عمليات على كل ال MenuItem المتضمنة فى ال MainMenu أو ال PopupMenu بالإشارة الى Items array فى ذلك ال object.

كما هو الحال فى جميع ال objects باتباع تقليد تسمية جيد هو السبيل الى إنشاء قوائم قابلة للحفظ، يختار Delphi تلقائياً أسماء ال MenuItems مثل File1، Open1، و Save2، والتي تعد ملائمة اخير هذه الاسماء مستخدماً الإرشادات التالية :

● من خلال ال menu-bar labels، لقد أضفت كلمة Menu لاسماء ال MenuItem - على سبيل المثال، FileMenu ل File menu object، و EditMenu ل Edit menu.

● للأوامر الخاصة فى pop-up menus، لقد جعلت الأوامر الجزء الأول من ال menu-bar label. على سبيل المثال، MenuItem command objects، فى قائمة ال File تسمى FileOpen، و FileSave، و FileSaveAs. وال Objects فى قائمة Edit لها أسماء مثل EditCut و EditCopy، و EditPaste. بعض تطبيقات Delphi تلحق Item الى هذه الاسماء مثل FileOpenItem، و EditCutItem. قد تريد ان تفعل مثلها، خاصة اذا كنت تخطط لاستخدام ال wizards لإنشاء application shells.

## الباب الخامس : بناء القوائم

و Objects مثل FileMenu (التي تمثل قائمة File) و OpenFileDialog (التي تمثل امر ال Open في قائمة File) تعد Objects من نفس ال MenuItem class ، كما هو في كل بنود القائمة - حتى separator bars التي تقسيم الاوامر الى فئات . حدد خاصية ال Caption لل MenuItem بالنص الذي تريد عرضه في menu bar أو الأمر . عند تصميم قائمة جديدة ، اجد من الاسهل ان اكتب كل ال Captions الخاصة بقوائمى ، ثم اغير ال Names الافتراضية الخاصة MenuItem objects ، ولكن يمكنك أن تدخل هذه القيم بأى ترتيب .

يمكنك ايضاً تعيين قيم ل MenuItem object اثناء التشغيل لإنشاء قوائم ديناميكية تدخل وتخدم الاوامر طبعاً لاحتياجات البرنامج . على سبيل المثال ، جرب هذا الخطوات :

١ - أضف ال MainMenu object على ال form واضغطها مرتين لفتح ال Menu Designer .

٢ - قم بإنشاء قائمة File أمر Exit (اكتب &File و E&xit لوضع خط تحت مفاتيح الاختصار x و F) .

٣ - اختر بند كل قائمة إما فى ال Menu Designer أو فى ال form ، قم بتغيير اسم الخاصية File1 ليصبح FileMenu . غير Exit1 ل FileExit .

لديك الآن ال MainMenu يمتلك إثنين من ال object ، ال MenuItem ، FileExit و FileMenu . يمكنك إدخال عبارات لبرمجة عناصر القوائم هذه لمجموعة من ال event handler . ثم ، أضف هذه العبارة بين البداية والنهاية :  
with FileExit do

Visible := not Visible;

وكذلك أدخل عبارة Close; ل FileExit command . قم بتشغيل البرنامج واضغط الزر لتشغيل وإبطال امر ال Exit . ان استخدام خاصية ال Visible يعد طريقة فعالة لإنشاء قوائم ديناميكية ذات اوامر تظهر وتختفى طبقاً لظروف البرنامج المختلفة .

أو ، يمكنك ابطال بند قائمة بتحديد خاصية ال Enabled لها ب True أو False على سبيل المثال ، قم بتغيير ال code السابق الى :

```
with FileExit do
  Enabled := not Enabled;
```

### محاكاة الأمر:

لتحاكي اختيار أمراً، إستدع أى MenuItem's Click method. قد يكون هذا مفيداً، مثلاً، عندما تريد أن يقوم الـ event handler بأداء نفس المهمة كأمر pop-up menu. إذا كان command object هذا يعتبر OptionsProject، يمكنك تنفيذ الـ event handler الخاص بالأمر بالعبارة التالية:

```
OptionsProject.Click; { Simulate selecting OptionsProject }
```

إن الضغط ليس له تأثير على الـ MenuItem object الذى يتصدر قائمة فى الـ window's menu bar رغم ذلك، لا يزال الـ MenuItem يتلقى الـ OnClick لأى نشاط يقوم به المستخدم مثل ضغط الفأرة على اسم بند قائمة. يمكنك استخدام هذا الـ event لأداء أعمال مثل تغيير اسماء الأوامر فى قائمة، أو إدخال علامات صح.

### خاصية الـ Items:

يمكنك أيضاً التمكن من الـ MenuItem objects بواسطة خاصية الـ Items، والتى تعتبر عنصراً فى الـ MainMenu، والـ PopupMenu، والـ MenuItem. وتعتبر الـ Items object من الـ TMenuItem التى يمكنك استخدامها لإستدعاء الـ Methods مثل الـ Items.Insert، أو الـ array فى تعبيرات مثل الـ MyMenu.Items[N]. فى الـ MainMenu object، تحتوى خاصية الـ Items على عناصر الـ pop-up menu الخاصة بالـ menu bar، على سبيل المثال، فى برنامج ذى ثلاث File-pop-up menus، و Edit، و Help- تقوم العبارات التالية بتغيير اسم القائمة الأولى الى Presto-Chango، وتبطل قائمة الـ Edit، وتجعل قائمة Help غير مرئية:

```
with MainMenu1 do
begin
  Items[0].Caption := 'Presto-Chango';
  Items[1].Enabled := False;
  Items[2].Visible := False;
end;
```

## الباب الخامس : بناء القوائم

لكل MenuItem object أيضاً خاصية Items، التي يمكنك استخدامها للتوصل الى أوامر القائمة، على سبيل المثال، لإبطال أمر ال FileExit، بفرض أن هذا هو الأمر الوحيد في القائمة، يمكنك استخدام هذه العبارة:

```
FileMenu.Items[0].Enabled := False;
```

وهذا له نفس التأثير الذي تحدثه العبارة التالية، والتي هي أبسط في أغلب الحالات:

```
FileExit.Enabled := False;
```

وكقاعدة عامة، لأداء مهام ال MenuItem objects، من الأسر أن تشير الى اسماء ال object، كما هو الحال في المثال السابق. لتحديد خصائص متعددة رغم ذلك، قد يكون من الأفضل أن تستخدم خاصية ال Items. أى من الأسلوبين يتوصل الى نفس MenuItem objects، لذا استخدم ما يحلو لك. على سبيل المثال، العبارة التالية تستخدم متغير Integer في for Loop لإبطال كل الأوامر في ال FileMenu:

```
for I := 0 to FileMenu.Count - 1 do  
  FileMenu.Items[I].Enabled := False;
```

وبالمناسبة، لأن العبارة السابقة تشير الى ال FileMenu مرتين، يمكنك تبسيط ال code باستخدام عبارة with. رغم أن النتيجة تحمل سطوراً إضافياً، ولكنها أكثر فعالية من الناحية المنطقية:

```
with FileMenu do  
  for I := 0 to Count - 1 do  
    Items[I].Enabled := False;
```

وعبارة ال with تخبر Delphi أن يستخدم ال Count وال Items التابعين لل FileMenu، مما لا يدفع الحاجة الى أن تسبق هذه الاسماء FileMenu ونقطه. وهذا يوفر الكتابة، ويمكن أن يساعد ال compiler على إنتاج code أكثر فعالية. حاول استخدام with قدر استطاعتك.

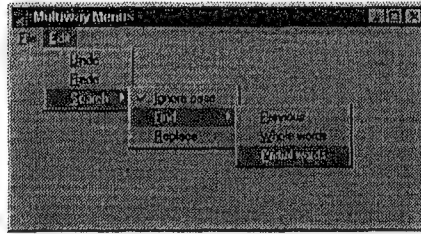
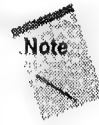
وخاصية ال Count لل FileMenu تساوى عدد ال MenuItem objects في ال Items array. ولأن array index الأول يساوى صفر، فإن object

index الأخير يساوى 1 - Count. ضع هذا فى الاعتبار دائماً عند كتابة loop للتعامل مع خاصية ال Items ك Items.

### :Multiway menus

يمكن لأى بند فى ال pop-up menu أن يكون له قائمة فرعية متصلة به . كما يوضح الشكل (٥-١)، يمكنك أن تحمل هذه الفكرة الى أقصى مدى، ولكن الأفضل ألا تذهب بعيداً. بالرغم من أن multiway menus تعتبر نافعة لتنظيم هياكل الأوامر المعقدة، ولكن يمكن أن تكون صعبة الاستخدام.

**ملحوظة:** يقوم Windows 95 بتنشيط إختيار القائمة بألا يطالب المستخدمين بضغط زر الفأرة عدة مرات. وهذا يجعل multiway menus أسهل فى الاستخدام أكثر منها فى النسخ السابقة من ال Windows. إن الإكتفاء بتكوين مستويان يعد من أفضل الأمور. ، تعتبر الثلاث مستويات دائماً عدد كبير جداً.



شكل (٥-١): Multiway menus توضح تداخل هياكل الأوامر المعقدة. هذا العرض من برنامج ال MultiMen على القرص المدمج المرفق بالكتاب

لإنشاء قوائم فرعية فى ال Menu Designer، اتبع الخطوات التالية:

- ١- قم بإنشاء أمر قائمة ك search [أنظر الشكل (٥-١)].
- ٢- قم بإبراز الأمر واضغط السهم الأيمن Ctrl+right. هذا يؤدي الى إنشاء قائمة فرعية يمكنك التعامل معها كما تفعل فى أى من pop-up menu windows أخرى.
- ٣- أعد هذه الخطوات لإنشاء مستويات تداخل إضافية.

## الباب الخامس : بناء القوائم

ولا تتطلب الـ Multiway menus برمجة خاصة . فإنك تنشئ event handlers لأوامرها بنفس الطريقة التي تفعلها في pop-up menus ذات المستوى الواحد . بعبارة أخرى ، تداخل القوائم أمر بصرى خالص . لذلك ، بصرف النظر عن مستوى التداخل ، يجب أن يكون لكل MenuItem objects في الـ MainMenu أو الـ PopUpMenu خصائص Name فريدة .

### :Menu item shortcut keys

يمكنك أن تصنع نوعين من الـ shortcut key يتم تحديدهما لأوامر القائمة . يمكنك أن تسبق الحرف بـ & لتصمم مفتاح Alt . على سبيل المثال ، إذا حددت الـ Caption الخاص بالـ MenuItem بـ &Configure ، يمكن للمستخدمين أن يضغطوا Alt+C لفتح قائمة الـ Configure .

والنوع الثاني من الـ shortcut key يسمى accelerator . من المتعارف عليه أن accelerator - عبارة عن مجموعة من مفاتيح عديدة أبجدية ، Shift ، Ctrl ، سهم ، و function keys وتسمى accelerators . لأن المستخدم يستطيع أن يضغطها لإختيار أوامر دون الحاجة لفتح نافذة قائمة .

لاتقم بإضافة الـ shortcut key labels في الـ MenuItem Captions . يقوم Delphi تلقائياً بإلحاق اسماء المفاتيح . على سبيل المثال ، إدخال E&xit للـ Caption الخاص بـ FileExit ، وحدد الـ shortcut key هذا الـ object بـ Alt+X . يقوم Delphi تلقائياً يعرض الأمر على أنه Exit Alt+X في الـ pop-up menu .

يمكنك اختيار الـ shortcut keys من قائمة الائحة التابعة للخاصية . ولكن ، لان هذه القائمة ليس لها مداخل عديدة ، اجد انه من الایسر ان تدخل الـ key label كنص . فمثلاً ، بدلاً من ان تضغط ALT + X ، إننى اكتب الخمسة رموز ALT + X في خاصية الـ ShortCut واضغط Enter . يجب ان تفعل هذا لتعيين مفاتيح مثل Enter والتي لا توجد في قائمة اللائحة - اكتب ENTER ، مثلاً . يمكنك ايضاً إنشاء اختصارات رموز بادخال A, B, C, 1, 9 أو أى مفتاح آخر تستطيع كتابته .

ويضع الجدول (٥-١) كشافاً بالـ standard accelerator shortcut key لبنود القائمة العامة . وتتبع تطبيقات الـ Windows 95 هذه المعايير أكثر منها في نسخ الـ Windows السابقة ، وإنها فكرة جيدة يجب ان تستخدمها في تطبيقاتك .

وهذه ليست قواعد جامدة، ان استخدام المفتاح يعتمد على التطبيق - فمثلاً، يمكن ان تستخدم مفتاح Ins لإدخال حقل بدلاً من ربط زر insert/overtyping الخاص بال editor. (إننى لا إتفق على ان ال File|Exit هو المعيار الرسمى لمفتاح ال Alt+F4. فالجميع يعرفون ان Alt+X هو accelerator standar للخروج من البرنامج).

جدول (٥-١) Standard Accelerator Shortcut Keys

Command	Menu	Shortcut
Cascade	Window	Shift+F5
Collapse all	Tree	Ctrl+* (keypad)
Collapse item	Tree	- (keypad minus)
Copy	Edit	Ctrl+C
Cut	Edit	Ctrl+X
Delete	Edit	Del
Exit	File	Alt+X (officially Alt+F4)
Expand all	Tree	* (keypad star)
Expand item	Tree	+ (keypad plus)
Expand/Collapse	Tree	Enter
Find	Edit	Ctrl+F
Find next	Edit	F3
Insert/Overtyping	Edit	Ins
New	File	Ctrl+N
Open	File	Ctrl+O
Paste	Edit	Ctrl+V
Print	File	Ctrl+P
Replace	Edit	Ctrl+H
Save	File	Ctrl+S
Search	Help	F1
Select all	Edit	Ctrl+A
Tile	Window	Shift+F4
Undo	Edit	Ctrl+Z

#### علامات الصح:

إن إضافة وإزالة علامات الصح من أوامر القائمة لن يكون سهلاً. ببساطة، حدد أى خاصية Checked لل MenuItem ب True لعرض علامة صح، أو ب

## الباب الخامس : بناء القوائم

False لمسح إحدى العلامات . يمكنك القيام بهذا أثناء وقت التصميم أو وقت التشغيل باستخدام عبارة مثل :

```
EditInsert.Checked := True;
```

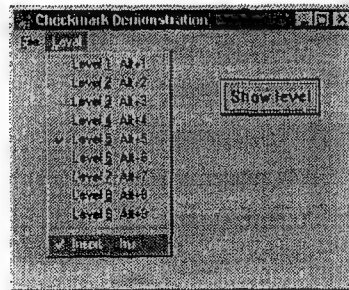
فى تصميم واجهة التطبيق للمستخدم الخاص بالبرنامج ، فكر جيداً اذا كنت سوف تجعل أمر ذو علامة صح يعمل ك toggle أو كإختيار من مجموعة من الأوامر المرتبطة بها . على سبيل المثال ، قد تعطى قائمة ال Level الخاصة بالألعاب مستويات صعوبة ، واحداً فقط هو الذى يمكن إختياره فى كل مرة . وأمر ال Insert فى قائمة ال Edit من المحتمل أن يكون لل toggle والحل .

قم بإنشاء toggle باستخدام عامل not لل Pascal كما فى هذه العبارة ،  
والتي قد تدخلها فى OnClick event handler الخاص بال MenuItem .  
with EditInsert do

```
Checked := not Checked;
```

فى أى مكان آخر فى البرنامج ، استخدم ال Checked فى عبارة if لإختيار عمل :  
if EditInsert.Checked then  
... { do something if menu item is checked }

قم بتشغيل ال Checks وجرب قائمة ال Level . ويستخدم البرنامج علامات صح بطريقتين : أن تختار من بين تسعة مستويات ، ليجعل أمر Insert من ال on الى off اضغط زر النافذة لعرض مواصفات المستوى الحالى . يوضح شكل (٢-٥) نافذة البرنامج .



شكل (٢-٥) : تعرض تطبيقات ال Checks كيفية استخدام علامات الصح ك toggle  
وتختار واحداً من مجموعة من الأوامر ، مثل المستويات التسع الموضحة هنا



توضح القائمة (٥-١) الـ form's source code والتي تظهر كيف يمكن أن تتشارك سلسلة من أوامر القائمة في نفس الـ event handler لإنشاء الـ LevelClick، عرفه كما هو موضح في TMainForm class، وأدخل عبارات الـ procedure في الـ implementation section يقوم الـ procedure بإغلاق علامات الصح للأوامر من Level1 إلى Level9، ثم يحدد خاصية الـ Checked للـ Sender بـ True. إن تعبير الـ TMenuItem(Sender) يجعل الـ Sender كـ TMenuItem object. لاستخدام الـ LevelClick، قم بتعيينه للـ OnClick event لكل MenuItem، من الـ Level1 إلى Level9.

القائمة (٥-١) Checks\Main.pas

unit Main;

interface

uses

Windows, SysUtils, Messages, Classes, Graphics, Controls,  
Forms, Dialogs, Menus, StdCtrls;

type

TMainForm = class(TForm)  
  MainMenu1: TMainMenu;  
  FileMenu: TMenuItem;  
  FileExit: TMenuItem;  
  LevelMenu: TMenuItem;  
  Level1: TMenuItem;  
  Level2: TMenuItem;  
  Level3: TMenuItem;  
  Level4: TMenuItem;  
  Level5: TMenuItem;  
  Level6: TMenuItem;  
  Level7: TMenuItem;  
  Level8: TMenuItem;  
  Level9: TMenuItem;

```
ShowButton: TButton;
N1: TMenuItem;
LevelInsert: TMenuItem;
procedure FileExitClick(Sender: TObject);
procedure LevelClick(Sender: TObject);
procedure ShowButtonClick(Sender: TObject);
procedure LevelInsertClick(Sender: TObject);
private
    { Private declarations }
function GetLevel: Integer;
public
    { Public declarations }
end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

const
    highLevel = 9; { Highest level command }

procedure TMainForm.FileExitClick(Sender: TObject);
begin
    Close;
end;

procedure TMainForm.LevelClick(Sender: TObject);
var
    I: Integer;
begin
```

```

with LevelMenu do
    for I := 0 to highLevel - 1 do
        items [I]. checked := fales;
        .. TMenuItem(Sender).Checked := True;
end;

function TMainForm.GetLevel: Integer;
var
    I: Integer;
begin
    GetLevel := -1;
    with LevelMenu do
        for I := 0 to highLevel - 1 do
            if Items[I].Checked then
                GetLevel := I + 1;
end;

procedure TMainForm.ShowButtonClick(Sender: TObject);
var
    S: string;
begin
    S := 'Level = ' + IntToStr(GetLevel) + ' - Insert: ';
    if LevelInsert.Checked
        then S := S + 'ON'
        else S := S + 'OFF';
    ShowMessage(S);
end;

procedure TMainForm.LevelInsertClick(Sender: TObject);
begin
    with Sender as TMenuItem do
        Checked := not Checked;
end;

```

end.

وتوضح Function GetLevel كيفية فحص ال MenuItem object من خلال Items array لتحديد أى من الاوامر له علامة صح . (1-) function اذا لم يكن هناك أمراً أشار إليه بعلامة صح ، أو انه يرجع المستوى من ١ الى ٩ .  
عندما تضغط الزر ، يقوم ال ShowButtonClick بإنشاء string يوضح المستوى الحالي (باستدعاء ال GetLevel function) والتحديد الحالى لل Toggle Insert . وال ShowMessage procedure الخاص ب Delphi يعرض النتيجة النهائية .

### : Floating Pop-up Menus

يوجد ثلاث إختلافات هامة فقط بين ال floating وال nonfloating pop-up menus وهى المكان ، المكان ، المكان . إنك تقوم ببرمجة ال PopupMenu objects بنفس الطريقة التى تبرمج بها ال MainMenu ، ولكن بدلاً من الظهور فى ال menu bar ، يمكن ان تظهر floating pop-up menu فى أى مكان من الشاشة . وتطبيق ال MemoPad ، من الباب الثانى ، يظهر كيفية انشاء واستخدام ال floating pop-up menu .

لإنشاء ال floating pop-up menu ، أضف ال PopupMenu component على ال form ، اضغط ال object مرتين ، أدخل الأوامر باستخدام ال Menu Designer بنفس الطريقة التى تتبعها مع ال MainMenu . قم بتعيين ال PopupMenu الخاصية ال PopupMenu بال form .

فى الغالب تتشارك اوامر ال floating pop-up menu و ال floating pop-up menu فى نفس event handlers كما فى القائمة الرئيسية للبرنامج ، ولكن هذه ليست قاعدة . يمكنك برمجة ال event handlers بشكل منفصل لل floating pop-up menu اذا اردت .

### زر الفأرة الأيمن:

لبرمجة الضغط لزر الفأرة الأيمن لعرض ال floating pop-up menu ، قم بتعيين اسم ال PopupMenu الخاصية ال PopupMenu بال form . تظهر القائمة عندما يضغط المستخدمون زر الفأرة الأيمن فى نافذة ال form .

يمكنك أيضاً تعيين ال PopupMenu فى وقت التشغيل ، والذي قد تفعله للتغيير بين اثنين أو أكثر من ال floating pop-up menu اعتماداً على الظروف المختلفة للبرنامج فى event handler للزر ، أو فى ال procedure لأمر القائمة ، استخدم عبارة مثل التالية لتعيين ال PopupMenu لخاصية ال PopupMenu التابعة لل form :

```
PopupMenu := PopupMenu2; { Assign PopupMenu2 to
PopupMenu }
```

### طرق أخرى لـ float:

استدع ال Popup method التابع لـ PopupMenu لعرض ال floating pop-up menu إستجابة لـ events مختلفة عن ضغط زر الفأرة الأيمن قم بتمرير إحداثيات الشاشة x و y لل Popup كما فى هذه العبارة :

```
PopupMenu1.Popup(100, 100);
```

إحداثيات ال x و ال y الخاصة بال Popup التى لها علامة بشاشة ال Windows ، ويكون (٠ ، ٠) الركن اليسر العلوى للشاشة .

عند تمرير إحداثيات ضغط الفأرة لل Popup ، لان هذه القيم لها علاقة بالنافذة ، استدع ClientToScreen function للتحويل الى الاحداثيات ذات العلاقة بالشاشة . أولاً قم بتعريف متغير من نوع ال TPoint

```
var
```

```
Pt: TPoint;
```

ثم ، فى MouseDown التابع لل form ، ادخل هذه العبارات :

```
Pt.X := X;
```

```
Pt.Y := Y;
```

```
Pt := ClientToScreen(Pt);
```

```
PopupMenu1.Popup(Pt.X, Pt.Y);
```

يعين أول سطرين إحداثيات الفأرة x و y اللذان تم تمريرهما ال event handler الى متغير ال TPoint . يستدع الخط الثالث ال ClientToScreen فى TControl class ، التى تعتبر واحدة من السلالات البعيدة لل TForm . وهذا

## الباب الخامس : بناء القوائم

يحول ال Pt من الاحداثيات المتصلة بال client-relative من النافذة الى القيم المتصلة بالشاشة . ويقوم السطر الأخير لتمرير قيم ال x و ال y المحولة ل method ال Popup التابعة ل PopupMenu .

### Dynamic Menus

يقدم Delphi تقنيات متنوعة لإنشاء قوائم تتغير بطريقة ديناميكية في وقت التشغيل . يمكنك ان :

- تغير menu bar كامل للنافذة .
  - تدخل وتحذف pop-up menu .
  - تضيف وتزيل الأوامر .
  - تدمج ال menu objects .
- توضح الفصول التالية هذه التقنيات وما يزيد .

#### تغيير القوائم

ان اسهل طريق لتغيير قائمة النافذة لتعيين MainMenu مختلف لخاصية Menu التابعة لل form أولاً ، إدخال إثنين أو أكثر من ال MainMenu ال form واستخدم ال event handler لإدخال اوامره . ثم ، ربما في برنامج Menu Designer لزر أو لأمر ، ادخل عبارة مثل :

```
Menu := MainMenu2;
```

عندما تقوم بتشغيل البرنامج ، يتغير ال menu bar من التعيين الخاص به في وقت التصميم ، الى MainMenu2 . يقوم لل menu bar تلقائياً بتغيير مظهره البصري عندما ينفذ البرنامج عبارة التعيين السابقة .

#### ادخال وحذف القوائم

يمكنك دمج ال menu object (انظر دمج MainMenu objects ، مؤخراً في هذا الباب) ولكن هناك اسلوباً أبسط يمكنك استخدامه لإدخال وحذف pop-up menus بأكملها في menu bar بالنافذة . لتجربة هذا الاسلوب ، أضفت MainMenu object منفرداً على ال form إضغط ال object مرتين ، واستخدم ال Menu Designer لإدخال قوائم وأوامر .

بعد ذلك ، قد بإنشاء ال `OnCreate` الخاص بال `form` ادخل عبارات مثل التالية لجعل القوائم القافزة غير مرئية :

```
OptionMenu.Visible := False;
WindowMenu.Visible := False;
```

أخيراً ، استجابة لضغط زر ما أو أمر قائمة (أو أى حدث آخر) ، حدد خاصية ال `Visible` لعنصر قائمة ب `True` . تظهر القائمة تلقائياً . وهذا يعد اسلوباً جيداً لبرمجة امر ال `Advanced Menus` السريع الذى يمد القوائم الى أقصى مدى لها . وتبقى بنود القوائم الغير مرئية نشطة ، ولا تزال تتلقى ال `OnClick events` . اذا كان العناصر القوائم هذه `shortcut key` مرتبطة بها ، فيمكن للمستخدمين اختيارها حتى اذا كانت الأوامر غير مرئية .

### تغيير عناصر القائمة:

لتغيير نص أمر ما ، ببساطة قم بتغيير `Caption` جديد لأى `MenuItem` `object` . على سبيل المثال ، يمكنك برمجة امر ال `Undo` لإختار المستخدمين بما يمكن إبطاله :

```
EditUndo.Caption := '&Undo deletion';
```

### إضافة، إدخال، وحذف عناصر القائمة:

يمكنك إضافة ، وإدخال ، وحذف اوامر من أى `pop-up menu` من `TMenuItem class` . على سبيل المثال ، يمكنك إلحاق اسماء ملفات الى نهاية `FileMenu object` بقائمة `File` . أو ، يمكنك استخدام هذا الاسلوب لتعديل اوامر `pop-up menu` فى وقت التشغيل .

والخطوة الأولى فى إضافة أو إدخال بند قائمة هى ان تنشئ ال `TMenuItem` `object` . أولاً ، قم بتعريف متغير مثل هذا الذى فى ال `event handler` :

```
var
  MI: TMenuItem;
begin
  ...
end;
```

## الباب الخامس : بناء القوائم

في كيان ال procedure، قم بإنشاء TMenuItem جديد باستدعاء ال Create method، وقم بتمرير ال object الاساسى، والذي يكون فى الغالب القائمة الأمر. لبند الأمر الجديد:

```
MI := TMenuItem.Create(FileMenu);
```

لديك الآن ال menu object جديد يُشار إليه ب MI. قم بتعيين ال caption الخاص بال object's والخصائص الأخرى له باستخدام عبارات مثل التالية:

```
MI.Caption := '&New command';
```

```
MI.Visible := True;
```

```
MI.ShortCut := ShortCut(vk_F1);
```

عندما تنتهى من تشكيل عنصر القائمة الجديد، أضف الى اسفل ال pop-up menu باستدعاء Add:

```
FileMenu.Add(MI); { Add menu item to bottom of File menu }
```

عند هذه النقطة، يفترض ال FileMenu object أن يحتوى ال MI، ويحذف هذا object فى الوقت المناسب. لا يجب عليك حذف الذاكرة المخصصة لل MI من جانب ال Create.

يمكنك استخدام MI مرة أخرى لإنشاء عنصر قائمة آخر، ولكن لكل مرة، يجب عليك إستدعاء ال Create method كما هو موضح هنا. على سبيل المثال، لا يمكنك تعيين caption مختلف لل MI وإضافته للقائمة. لوضعى برامج Pascal الخبراء: يعتبر ال MI مشيراً الى object تم تخصيصه بطريقة ديناميكية. فى البرامج، لا يجب عليك إبطال مرجعية هذا ال Pointer- ببساطة استخدمه كما كنت تستخدم متغير من نفس النوع.

أو، يمكنك إضافة بند قائمة بين عناصر أخرى. افترض ان لديك خط فاصل يسمى FileSep. لإدخال MI فوق هذا الخط، أولاً حدد موضع فهرس الفاصل وقم بتعيينه لمتغير Integer:

```
Index := FileMenu.IndexOf(FileSep);
```

ثم، بدلاً من استدعاء ال Add، ادخل MI باستدعاء ال Insert method وقم بتمريرها على قيمة ال Index وشكل القائمة الجديد:



```
FileMenu.Insert(Index, MI); { Insert menu item at Index }
```

يمكنك أيضاً حذف عناصر القائمة بواسطة قيمة الفهرس . استدع Delete  
ال method الخاص بال TMenuItem:

```
FileMenu.Delete(Index);
```

أو، يمكنك حذف بند قائمة باستدعاء Remove . على سبيل المثال، هذا  
يحذف أمر ال Exit من قائمة ال File:

```
FileMenu.Remove(FileExit);
```

يمكنك تعديل محتويات قائمة رئيسية أو pop-up menu من خلال خاصية  
ال Items . استخدم ال Items ك array من TMenuItem object . على سبيل  
المثال، MyMenu.Items[I] تشير الى ال TMenuItem للـ index [I] فى  
MyMenu .

### تعيين code لـ menu item:

عند إنشاء عنصر قائمة جديد، يجب عليك أيضاً تعيينه الى event handler  
لأداء عمل عندما يختار المستخدمون الأمر . وهذا يأخذ ثلاث خطوات:

١- قم بتعريف ال event handler procedure و ال form's class .

٢- قم بتنفيذ ال procedure فى implementation section .

٣- قم بتعيين ال procedure لـ OnClick الخاص بعنصر القائمة .

على سبيل المثال، اضع التعريف التالى TForm1 class (فى الغالب  
أضع هذا التعريف وأى تعريف آخر فى قطاع ال public أو ال private للـ class  
فى هذه الحالة، ضعه فى ال private):

```
procedure NewCommandClick(Sender: TObject);
```

بعد ذلك، قم بتنفيذ ال procedure فى أى مكان بقطاع ال

: implementation

```
procedure TForm1.NewCommandClick(Sender: TObject);
begin
  ShowMessage('New command executed!');
end;
```

## الباب الخامس: بناء القوائم

لاحظ ان تعريف الـ procedure كـ method في قطاع implementation باستخدام الـ class name، ونقطة، واسم الـ procedure بهذه الطريقة، ليصبح الـ method تابعاً للـ form object الذي تم إنشائه باستخدام الـ TForm1 class. أخيراً، قم بتعريف متغير TMenuItem مثل MI، قم بإنشاء الـ object، عين الخصائص التي تريدها، وعين الـ event handler للـ onClick event:

```
MI := TMenuItem.Create(FileMenu);  
MI.Caption := '&New command';  
MI.OnClick := NewCommandClick;
```

ادخل أو أضف الـ object الجديد في القائمة. عندما يختاره المستخدم، يستدع البرنامج الـ NewCommandClick.

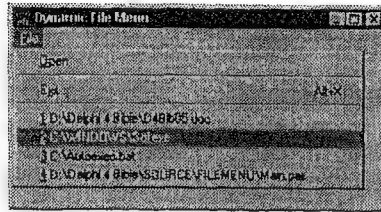
بالطبع، يمكنك تعيين الـ event handler لأمر قائمة موجود بالفعل الى الـ onClick event الخاص بـ قائمة جديد. فقط استخدم السطر الأخير من المثال السابق، وقم بتغيير NewCommandClick الى اسم الـ procedure الموجود بالفعل.

### إضافة أسماء ملف لقائمة ملف:

لا يعتبر الـ MenuItem objects أمراً. فيمكن ان يكونوا لإعطاء المعلومات ايضاً-على سبيل المثال، أسماء الملفات التي تم فتحها مؤخراً. يمكنك استخدام الـ Add، Insert، و Delete للتحكم في هذا الكشف - فقط قم بإنشاء الـ TMenuItem objects وعين أسماء ملف لخصائص الـ Caption. رغم ذلك، إنني أجد من الأيسر إدخال اوامر في قائمة File وتعيين أسماء ملف لها بمجرد ان يفتح المستخدمون ملفات جديدة.

**على القرص المدمج:** ان تطبيق الـ FileMenu الموجود على القرص المدمج المرفق بهذا الكتاب في دليل الـ Source\FileMenu يوضح الـ code الضرورية. يوضح شكل (٥-٣) قائمة الـ File للبرنامج مع اربعة ملفات تم فتحها حديثاً. يمكنك اختيار تلك الاسماء مثل أي أمر آخر، ولكل منها مفتاح Alt+N حيث ان الـ N هو الرقم الذي تحته خطأ. ان الـ code التي تتحكم في كشف اسم الملف تعتبر محيرة الى حد ما، كما توضح القائمة (٥-٢).





شكل (٣-٥): تقنية واجهة تطبيق عامة تضع  
كشفاً بالملفات التي تم فتحها حديثاً

في قائمة الـ File. قد تكون البرمجة محيرة، كما يوضح تطبيق الـ File menu.  
القائمة (٣-٥): T Filemenu\Main.pas;

unit Main;

interface

uses

Windows, SysUtils, Messages, Classes, Graphics, Controls,  
Forms, Dialogs, StdCtrls, Menus, Buttons;

type

TMainForm = class(TForm)

MainMenu1: TMainMenu;

FileMenu: TMenuItem;

FileExit: TMenuItem;

OpenButton: TButton;

OpenDialog: TOpenDialog;

FileOpen: TMenuItem;

FileSep1: TMenuItem;

FileSep2: TMenuItem;

FileName1: TMenuItem;

FileName2: TMenuItem;

FileName3: TMenuItem;

FileName4: TMenuItem;

```
BitBtn1: TBitBtn;
    procedure FileExitClick(Sender: TObject);
    procedure OpenButtonClick(Sender: TObject);
    procedure FileName1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FileExitClick(Sender: TObject);
begin
    Close;
end;

{ - Prompt for filename and add name to File menu }
procedure TMainForm.OpenButtonClick(Sender: TObject);
var
    S: string;
    I, K: Integer;
begin
    if OpenFileDialog.Execute then with FileMenu do
        begin
            if not FileSep2.Visible then
                FileSep2.Visible := True; { Make separator visible }
            K := IndexOf(FileName1);
```

```

for I := Count - 1 downto K + 1 do
begin { Move current filenames down one position
}
    S := Items[I - 1].Caption;
    S[2] := Chr(Ord('0') + (I - K + 1)); {
Alt-Shortcut }
    Items[I].Caption := S;
    Items[I].Visible := Items[I - 1].Visible;
end;
    FileName1.Caption := '&1 ' +
OpenDialog.FileName;
    FileName1.Visible := True;
    ShowMessage('Adding: ' + OpenDialog.FileName);
end;
end;

{- Get filename selected from File menu }
procedure TMainForm.FileName1Click(Sender: TObject);
var
    Filename: string;
begin
    with Sender as TMenuItem do
    begin
        Filename := Caption;
        System.Delete(Filename, 1, 2);
        end;
        ShowMessage('Selected: ' + Filename);
    end;

end.

```

**فكرة:** يوضح الـ `FileName1Click` فى القائمة (٥-٢). كيفية حل التداخل الذى قد يسبب لك مشكلة من وقت إلى آخر. ان الـ `Pascal Delete procedure` يستدعى لإزالة `Alt key`

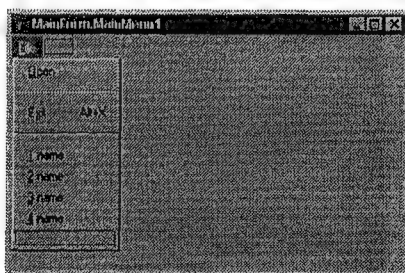
Tip

## الباب الخامس : بناء القوائم

labels التي تحتها خطأ المضافة لكل اسم ملف . رغم ذلك ، تخبر عبارة with السابقة Delphi ان يستخدم اعضاء ال TMenuItem ، واحداً منها يدعى Delete ايضاً . لقد وجدت حلاً لتداخل الاسماء هذا بأن سبقت Delete بـ System ونقطة ، والتي تقول للـ compiler ان يستخدم Delete string procedure الموجود في ال System unit ، وليس TMenuItem method الذي يحمل نفس الاسم . والاسماء المشتركة الواسعة الاستخدام الأخرى والتي قد تحتاج إلى تأهيلها تتضمن Insert ، Text ، Assign ، و Close .

اتبع هذه الخطوات لإنشاء قائمة File وإضافة اسماء الملفات التي تم فتحها حديثاً في اسفل نافذة القائمة :

١- أضف Menu component على ال form واستخدم ال Menu Designer لإنشاء قائمة File ذات فاصل (أضف هذه العلامة (-) في خاصية ال Caption) واربعة اسماء ملف وهمية لعناصر القائمة . انظر شكل (٥-٤) كمثال . إجعل اسم القائمة FileMenu ، واسماء الملفات الوهمية FileName1 ، FileName2 ، FileName3 ، FileName4 . إجعل اسماء الفواصل FileSep1 و FileSep2 . جدول (٥-٢) يعطى اسماء وخصائص ال component الهامة .



شكل (٥-٤) : قائمة تطبيق ال FileMenu التي تم فتحها في ال Menu Designer الخاص بـ Delphi ، مع اربعة عناصر اسماء ملفات وهمية

٢- حدد ال Visible بـ False للفاصل وكل بنود القائمة الوهمية . ان البرنامج يجعلها مرئية عندما يفتح المستخدمون ملفات جديدة .

٣- قم بإنشاء ال event handler للمدخل الوهمي الأول باختياره في ال Menu Designer . انظر FileName1Click في القائمة ، والتي تعرض اسم الملف المختار . عين نفس هذا ال event handler إلى البنود الوهمية الأخرى .

٤- إضف ال object OpenFileDialog على ال form .

٥- قم بإنشاء ال event handler لأمر ال Open أو الزر ما ، وأضف البرمجة من ال OpenFileDialog procedure .

### استخدم shortcut keys :

ان Menu unit الخاصة بـ Delphi ، مضافة إلى أى برنامج مع MainMenu objects أو PopupMenu ، تقدم اربعة نظم فرعية تفيد فى تعيين استخدام accelerator shortcut keys . للحصول عليها ، أضف MainMenu object على ال form ، واستخدم ال Menu Designer لإنشاء قائمة ذات عنصراً واحداً على الأقل يسمى Demo . إننى استخدم الاسم الافتراضى ، هنا وهو Demo1 .

جدول (٥-٢) : خصائص ال FileMenu Application Component

Component	Name	Property	Value
Form	MainForm	Caption	Dynamic File Menu
Button	OpenButton	Caption	Open file
BitBtn	BitBtn1	Kind	bkClose
OpenDialog	OpenDialog	Filter	All files (*.*) *.*
MainMenu	MainMenu1		
MenuItem	FileOpen	Caption	&Open
MenuItem	FileExit	Caption	E&xit
MenuItem	FileSep1	Caption	-
Component	Name	Property	Value
MenuItem	FileSep2	Caption	-
		Visible	False
MenuItem	FileName1	Caption	&1 name
		Visible	False
MenuItem	FileName2	Caption	&2 name
		Visible	False
MenuItem	FileName3	Caption	&3 name
		Visible	False
MenuItem	FileName4	Caption	&4 name
		Visible	False

استخدم ال ShortCut function لتعيين accelerator key لعنصر قائمة .

يعرف Delphi ال function كما يلى :

## الباب الخامس : بناء القوائم

function ShortCut(Key: Word; Shift: TShiftState): TShortCut;  
باستخدام ShortCut ، يمكنك تعيين F9 الخاصية ال ShortCut الخاص  
بـ Demo1 (للاستمرار ، اضع هذه العبارة لبرنامج ال event handler الخاص بالزر) :

Demo1.ShortCut := ShortCut(vk\_F9, []);

يعرض Delphi بصورة تلقائية ال shortcut keys في pop-up menu إلى اليمين من أمر ال Demo . لا يجب عليك تعديل ال caption الخاص ببند القائمة .  
تعرف الاقواس الفارغة ال null set ل shift keys لتعيين Alt+F9 ، حدد  
قيمة TShiftState في الاقواس :

Demo1.ShortCut := ShortCut(vk\_F9, [ssAlt]);

لان shift state تعتبر Pascal set ، يمكنك تحديد قيم متعددة منفصلة عن  
طريق الفصله . على سبيل المثال ، هذه العبارة تعيين ال Ctrl+Shift+A كاختصار  
(لا يهم ترتيب القيم داخل المجموعة) :

Demo1.ShortCut := ShortCut(Ord('A'), [ssCtrl, ssShift]);

لا يمكنك تحديد مفتاح ال A باستخدام ال virtual key code vk\_A ،  
بالرغم من ان هذه وغيرها من ال codes الرقمية الابدجية الواردة في online  
الخاص بـ Delphi وكذلك ، ملف Windows.pas source code في دليل  
Source\RTL الخاص بـ Delphi . استخدم ال Ord function الخاصة بـ Pascal  
ورمز حرفي بين علامتي تنصيص فردية لتخصيص مفاتيح رموز ال ASCII .  
لتقسيم shortcut key إلى قيمة منفصلة ، استدع ال ShortCutToKey ،  
الذي يتم تعريفه بالآتي :

procedure ShortCutToKey(ShortCut: TShortCut;

var Key: Word; var Shift: TShiftState);

قم بتمرير خاصية ال ShortCut الخاصة ببند القائمة إلى ال parameter  
الأول . يعين ال procedure ، ShortCut's virtual key الخاص بال ShortCut's  
لل Key والمجموعة التابعة لها من قيم ال shift-state .



وتقوم function أخرى، وهى TextToShortcut، بتحويل قيمة ال string إلى shortcut key. على سبيل المثال، خصص Alt+X بهذه العبارة:

```
Demo1.ShortCut := TextToShortcut('Alt+X');
```

يمكنك استخدام هذا الأسلوب لحث المستخدمين على إضافة shortcut key، ربما فى configuration utility تسمح للمستخدمين ببرمجة أوامر القوائم الخاصة بهم:

```
var
  S: string;
begin
  S := InputBox('Input Dialog', 'Enter shortcut key', '');
  if Length(S) > 0 then
    Demo1.ShortCut := TextToShortcut(S);
end;
```

تعتبر وظيفة ال InputBox فى متناول اليد لحث المستخدمين على عمل قيم ال string وال string الأول هو dialog's caption، والثانى هو استفسار معروض فى نافذة ال dialog والثالث (هنا بـ null) هو قيمة ال string الافتراضية وتعود القيمة بذلك اذا لم يدخل المستخدم قيمة ال string.

قم باداء العملية العكسية مع ال ShortCutToText، التى تعرف بالآتى:

```
function ShortCutToText(ShortCut: TShortcut): string;
```

على سبيل المثال، يمكنك عرض shortcut key لعنصر قائمة ك string فى dialog box:

```
ShowMessage('Demo1 shortcut = ' +
  ShortCutToText(Demo1.ShortCut));
```

### تشغيل وإبطال الأوامر:

لقد شرحت بالفعل فى هذا الباب كيفية تشغيل وإبطال عناصر القائمة بتحديد خاصية ال Enabled بـ True أو False. عندما تكون ال Enabled محددة بـ False، يكون ال caption الخاص ببند القائمة معتم فى نافذة ال pop-up menu،

## الباب الخامس : بناء القوائم

ولا يمكن للمستخدمين إختياره من الناحية العملية ، ان إعتام الاوامر الصحيحة في الأوقات السليمة يتطلب تخطيطاً دقيقاً .

واسهل الطرق هو كتابة procedure بشكل قائمة بأكملها طبقاً لـ global flags والقيم الأخرى . على سبيل المثال ، قم بتعريف متغير Boolean FileIsOpen ، واكتب الـ procedure لتشغيل أو ابطال أوامر الـ Save والـ Open والأوامر الأخرى التابعة لقائمة File طبقاً لقيمة المتغير .

**على القرص المدمج:** ان تطبيق الـ ToDo على القرص المرفق بهذا الكتاب في دليل الـ Source\ToDo يوضح كيفية عمل هذا لقائمة File عامة . والـ form class الخاصة بالبرنامج ، وهي TMainForm ،



تعرف procedure يسمى EnableMenu .:

```
procedure EnableMenu;
```

وينفذ البرنامج الـ procedure كما هو موضح في شكل (٥-٣) . وتقوم العبارات بتشغيل وإبطال الاوامر في قائمة File الخاصة بالبرنامج طبقاً لحالات حفظ الملف وفتح الملف التابعة للبرنامج ان نص هذه القائمة موجود في ملف Main.pas الخاص بمشروع ToDo على القرص المدمج .

**القائمة (٥-٣):** الـ EnableMenu procedure من عينة الـ ToDo

```
applicationprocedure TMainForm.EnableMenu;
```

```
var
```

```
  I: Integer;
```

```
begin
```

```
  with FileMenu do
```

```
    begin
```

```
      for I := 0 to Count - 1 do      { Enable File commands }
```

```
        Items[I].Enabled := True;
```

```
        if not FileDirty then
```

```
          begin { No edits }
```

```
            FileSave.Enabled := False;    { Must use Save as }
```

```
            if Length(Filename) = 0 then  { i.e. file not named }
```

```
begin { No edits; no name }
    FileSaveAs.Enabled := False; { Nothing to save }
    FilePrint.Enabled := False; { Nothing to print }
end;
end;
end;
end;
```

برنامج ال ToDo يستخدم flag وهو ال FileDirty للإشارة اذا ما كانت هناك تغييرات قد حدثت للـ ملف مفتوح. ويستخدم ال EnableMenu procedure هذا ال flag لتشغيل امر ال Save الخاص بالقائمة، والذي يظل معتمداً (وبذلك يبقى دون اختيار) اذا كانت بيانات الملف لم تتغير. ويقوم ال procedure ايضاً بتشغيل وإبطال اوامر اخرى مثل Print و Save As.

لاستخدام ال EnableMenu، قم بإنشاء event handler للـ OnClick، event الخاص بالـ FileMenu، استدع ال procedure. كنتيجة لذلك، يتم تشكيل اوامر القائمة قبل ان تصبح نافذة pop-up menu مرئية. على سبيل المثال، ها هو event handler للـ OnClick من ال ToDo:

```
procedure TMainForm.FileMenuClick(Sender: TObject);
begin
    EnableMenu; { Enable/disable commands before menu opens }
end;
```

فى تطبيق أكثر تعقيداً، قد يكون من الأفضل ان تكتب procedure لتشغيل قائمة لكل من قوائم البرنامج، File، Edit، و Window وغيرها. استدع ال procedures من ال OnClick لكل بند MainMenu. رغم ذلك، هذا لا يمنع المستخدمين من اختيار اوامر بضغط shortcut key. والسبيل الأكثر اماناً هو ان يؤدي كل امر الصلاحيات الخاصة به. على سبيل المثال، امر ال FileSave قد يخرج ببساطة دون اتخاذ أى عمل اذا لم يكن هناك ملفاً مفتوحاً، وبذلك، لا يوجد شئ يراد حفظه.

### دمج وفصل MainMenu objects

يمكن لأي form ان يكون لها MainMenu objects متعددة والتي يمكنك دمجها في بعضها لإنشاء قوائم ديناميكية . وتعتبر هذه التقنية مفيدة بالأخص في البرامج ذات two forms أو أكثر . عندما يعرض البرنامج form ثانوية ، تنتج أوامر ال MainMenu الخاصة بها تلقائياً مع النافذة الرئيسية للبرنامج . على سبيل المثال ، يمكن لل configuration dialog أن يستخدم هذه التقنية لدمج مجموعة جديدة من الأوامر في menu bar . عندما يغلق المستخدم ال dialog تختفى الأوامر المدرجة بصورة تلقائية .

### دمج MainMenu objects:

إن العنصر الرئيسى فى القوائم المدرجة هو قيمة ال GroupIndex Byte فى ال TMenuItem class . وهذه القيمة ، والتي قد تتراوح من صفر إلى ٢٥٥ ، تحدد تأثير الدمج . استخدم قيم GroupIndex المساوية لإحلال إحدى القوائم مكان أخرى . استخدم قيم أقل لإدخال قوائم إلى اليسار . استخدم قيم أعلى لإدخال قوائم إلى اليمين .

قم بتعيين قيم GroupIndex فى مضاعفات ال ١٠ لتيسير عملية إدخال قوائم جديدة بين القوائم الموجودة . ولكن ، فى تطبيقات ال OLE (أنظر الباب السادس عشر) ، يتطلب ال Windows منك أن تستخدم قيم GroupIndex محددة لتشغيل دمج القوائم in-place editing .

اتبع هذه الخطوات لدمج MainMenu objects فى نفس ال form ولتجربة هذه التقنية :

١ - أضف ال two MainMenu objects على ال form إننى استخدم اسماء ال object الافتراضية وهى MainMenu1 و MainMenu2 ، ولكن يمكنك تعيين هذه الاسماء اذا أردت .

٢ - اضغط مرتين ال MainMenu1 واستخدم ال Menu Designer لإنشاء قائمة Demo ذات أمر ال Advanced . يقوم Delphi بتسمية عنصر القائمة بـ Demo1 طبقاً للنظام الافتراضى . اختر Demo1 فى نافذة ال Object

Inspector، وحدد قيمة الـ GroupIndex الخاصة بـ Demo1 بصفر (القيمة الافتراضية).

٣- اضغط مرتين MainMenu2، وقم بإنشاء قائمة Extra ذات أمر Exit. يقوم Delphi بتسمية بند القائمة بـ Extra1 طبقاً للنظام الافتراضي. اختر Extra1 في نافذة الـ Object Inspector، وحدد قيمة الـ GroupIndex الخاصة بـ Extra1 بواحد.

٤- قم بإنشاء الـ event handler للـ Advanced command بإختيار في الـ form أو في الـ Menu Designer، وأدخل العبارة التالية بين البداية والنهاية:

MainMenu1.Merge(MainMenu2);

٥- قم بإنشاء الـ event handler لأمر الـ Exit. (لأن الـ MainMenu2 غير ظاهرة في الـ form، يجب أن تستخدم الـ Menu Designer لإختيار الأمر). أدخل عبارة Close للـ procedure الخاص بهذا الأمر.

٦- اضغط F9 لتشغيل البرنامج. اختر أمر الـ Demo1Advanced لدمج القائمة الثانوية، وجعل أمر الـ Exit متاحاً.

قم بتجربة قيم GroupIndex مختلفة. على سبيل المثال، حدد الـ GroupIndex بواحد لكل من Demo1 و Exit1. إن إختيار أمر الـ Advanced عندئذ يجعل قائمة الـ Extra تحمل محل الـ Demo1. أو، حدد الـ GroupIndex بقيمة أعلى في الـ Demo1 منها في الـ Exit1. يقوم أمر الـ Advanced الآن بإدخال قائمة الـ Extra إلى اليسار من Demo1.

### فصل الـ MainMenu objects:

قم بإبطال تأثير الدمج بإستدعاء الـ UnMerge method لأي MainMenu object. على سبيل المثال، أضف زرأ على نافذة الـ form واستخدم العبارة التالية في الـ OnClick الخاص بالزر. إن ضغط الزر يحذف الـ MainMenu2 من الـ MainMenu1:

MainMenu1.UnMerge(MainMenu2);

### دمج MainMenu objects في forms متعددة:

عند دمج MainMenu objects في two forms أو أكثر، استخدم خاصية الـ AutoMerge لتحديد ما إذا كان الدمج يتم بصورة تلقائية أو يحدث بموجب تحكم البرنامج. قم بتجربة هذه الخطوات لتجربة دمج قوائم forms متعددة:

١- أضف MainMenu object في الـ form، اضغطه مرتين، استخدم الـ Menu Designer لإنشاء قائمة Demo ذات أمر واحد، وهو الـ Show form. مازالنا في الـ Menu Designer، اضغط مرة واحدة قائمة Demo، ثم استخدم الـ Object Inspector لتحديد الـ GroupIndex بصفر (القيمة الافتراضية). لاحظ أن Menu object يسمى Demo1 وهو object من الـ TMenuItem class. تأكد من إختيار الـ menu objects باستخدام الـ Menu Designer، وليست الـ form التي تظهر بها القائمة.

٢- أضف form خالية ثانية إلى المشروع باستخدام أمر الـ File/New Form. قم بإعادة تحديد حجم نافذة هذه الـ form، والتي تسمى Form2، وحركها جانباً.

٣- في الـ Form2، أضف الـ MainMenu object منفصل، وحدد خاصية الـ AutoMerge التابعة له بـ True. لاحظ أن الـ object يسمى MainMenu1، مثل MainMenu object الآخر. ولأن الـ objects تقع في forms منفصلة، لا يوجد تداخل بين الاسماء. اضغط مرتين الـ MainMenu1 object التابع لـ Form2 واستخدم الـ Menu Designer لإنشاء قائمة Advanced ذات أمر Close. اختر هذا الأمر وأدخل عبارة Close في الـ event handler.

٤- وبينما مازالنا في الـ Form2 ونستخدم الـ Menu Designer، اضغط مرة واحدة بند قائمة الـ Advanced وحدد خاصية الـ GroupIndex الخاصة بها بواحد. (إذا لزم الأمر، اضغط مرتين الـ MainMenu1 object في الـ Form2 للرجوع للـ Menu Designer). إنتقل إلى الـ Form1، اختر أمر الـ Demo/Show form، وأدخل هذه العبارة في الـ event handler الناتج.

Form2.Show;

٥- أضف تعريف ال uses تحت الكلمة الرئيسية Unit1 implementation

مباشرة:

uses Unit2;

٦- افعل نفس الشيء لـ Unit2 ، ولكن قم بالإشارة إلى Unit1 بالآتى (كل unit تستخدم الأخرى حالياً):

uses Unit1;

٧- أخيراً، قم بإنشاء ال event handler للـ OnClose event الخاص بـ Form2. أدخل هذه العبارة فى ال procedure.

Form1.MainMenu1.UnMerge(MainMenu1);

٨- اضغط F9 لتشغيل البرنامج. اختر أمر ال DemoShow form ، ولاحظ ان النافذة الرئيسية لها الآن عنصر قائمة Advanced. اختر أمر ال Close الخاص بهذه القائمة لإغلاق النافذة الثانوية واستعادة القائمة المدمجة فى تنسيقها الأصلية.

قد تبدو الخطوات السابقة معقدة بعض الشيء فى المرة الأولى التى تقوم بها. ولكن هذه التقنية مفيدة عندما يملئ عليك مظهر نافذة مستوى فرعى تغييراً لمجموعة القوائم الخاصة بالبرنامج- على سبيل المثال، عند برمجة نافذة Find dialog. عند دمج MainMenu objects فى forms متعددة، تذكر هذه الأفكار:

- أضف MainMenu object واحداً على كل form. حدد ال AutoMerge بـ False للـ main window's object. حدد ال AutoMerge بـ True لكل MainMenu object الأخرى. اذا كنت لا تريد دمجاً بصورة بالقائمة، حدد ال AutoMerge بـ False لكل القوائم، واستدع Merge لأداء دمج القوائم كما فعلت فى حالة الدمج فى form واحدة.

- أضف اسماء unit module ثانوية (Unit2، Unit3، وما إلى ذلك) إلى القطاع uses فى Unit1. وهذا يخبر ال module الرئيسية ان تستخدم ال modules الثانوية.

- أضف اسم ال main module الرئيسية (Unit1، مثلاً) لكل قطاع uses جديد فى كل modul ثانوية.

## الباب الخامس : بناء القوائم

● استدع `UnMerge` للـ `MainMenu object` الخاص بالـ `form` الرئيسية لإزالة قائمة بصورة تلقائية عند إغلاق الـ `form` الثانوية . انظر الخطوة رقم ٧ في البرنامج التعليمي السابق . بالرغم من تحديد الـ `AutoMerge` ، لا يمكن ان يكون الفصل بصورة تلقائية عند إغلاق الـ `form` (رغم انه ربما كان يجب ان يكون كذلك) .

● استدع الـ `Show` لعرض نافذة الـ `form` الثانوية الـ `child` أو الـ `modeless dialog` للنافذة الرئيسية الأم .

● و `Delphi` يجعل اسم كل `MainMenu object` في كل `form` ، `MainMenu1` . للحصول على برنامج اوضح ، اختر اسماء افضل لهذه الـ `objects` مثل `MainFormMainMenu` و `FindFormMainMenu` .

تتطلب تطبيقات الـ `OLE` والـ `MDI` تقنيات مختلفة لدمج القوائم . انظر الباب الخامس عشر والسادس عشر لمزيد من المعلومات حول برمجة الـ `OLE` والـ `MDI` .

### تعديل قائمة النظام:

تظهر قائمة النظام عندما تضغط ايقونة البرنامج في الركن الايسر العلوى بالنافذة . لا يجب ان تحتاج معظم التطبيقات إلى إضافة اوامر لهذه القائمة - وكذلك ، يتعامل `Delphi` بصورة تلقائية مع الاوامر المعيارية للقائمة . ولكن ، اذا اردت تعديل قائمة النظام لتطبيق معين ، فنحن نشرح لك كيفية ذلك .

تعتبر قائمة النظام ملكاً للـ `Windows` ، وليس للتطبيق ولذلك ، لا يمكنك استخدام `MainMenu object` للتوصل إلى قائمة النظام . لإضافة أمر ، استدع الـ `Windows AppendMenu` . للاستجابة إلى اختيار الأمر ، اضع `message-handler procedure` للـ `form's class` . اتبع هذه الخطوات لتجربة التقنية واطفء أمر الـ `About...` لقائمة النظام :

١ - قم بإنشاء الـ `handler` للـ `OnCreate event` الخاص بالـ `from` الرئيسية . ادخل البرمجة للـ `FormCreate` من القائمة (٥-٤) " هذا النص موجود على القرص المدمج في ملف `Main.pas` لمشروع الـ `SysMenu` ) .



٢- قم بتعريف رقم ثابت ليمثل الأمر الجديد . استخدم أى قيمة صغيرة مثل الكسر السداس \$00A0 (١٠٠ عشر) . يستخدم ال Windows البت الرابع القلائل لهذه القيمة للـ command flag والتي يجب ان تكون صفراً . واسهل طريقة لتلبية هذا المتطلب هى تعيين قيم . فى الكسر السداسى مع آخر رقم مساوياً لصفر . على سبيل المثال ، اصف هذا التعريف للـ unit's implementation :

const

cm\_About = \$00A0;

٣- فى ال form's class (وهى TForm1 طبقاً لنظام الافتراضى) ، ادخل تعريف ال message-handler التالى فى ال class's private . والسطر الأول يعرف ال procedure ، والذي يجب ان يكون به ال procedure المتغير الفردى TWMSysCommand كما هو موضح . والسطر الثانى يخبر Delphi ان تستدعى هذا Delphi عندما تتلقى ال form رسالة wm\_Syscommand من ال Windows . تشير هذه الرسالة إلى ان المستخدم قد إختار أمر قائمة نظام :

procedure WMSysCommand(var Message: TWMSysCommand);  
message wm\_SysCommand;

٤- قم بتنفيذ ال WMSysCommand كما هو موضح فى القائمة (٥-٤) . قم بتشغيل البرنامج واختر أمر ال About... الخاص بقائمة النظام لعرض display Box . فى القائمة (٥-٤) ، تظهر ال WMSysCommand كيفية التعامل مع رسالة فى ال form class . يتلقى ال procedure متغير TWMSysCommand الذى يصف محتويات الرسالة . فى هذا المتغير ، ليمثل مجال ال CmdType قيمة الامر الذى تم اختياره . اذا كانت هذه القيمة تساوى ثابت ال cm\_About الخاص بنا ، تستدعى عبارة case ال ShowMessage لعرض dialog box . لأداء الانتاج الافتراضى للأوامر الاخرى لقائمة النظام ، يستدعى ال procedure inherited لقيم ال unhandled CmdType .

ويعرض ال OnCreate الخاص بال form كيفية إيجاد ال window handle لقائمة النظام . يخزن ال procedure هذا ال handle فى متغير MenuH ، من نوع ال HMenu (menu handle) . أولاً ال GetSystemMenu يعيد ال

## الباب الخامس : بناء القوائم

handle لنافذة ال form - قيمة ال Handle المتغيره فى هذه العبارة تنتمى لل form . وتضيف Windows AppendMenu function أمر أمر ال About... لهذه القائمة .

القائمة (٤-٥) : إضافة اوامر لقائمة نظام التطبيق

```
procedure TForm1.WMSysCommand(var Message: TWMSysCommand);
begin
  case Message.CmdType of
    cm_About: ShowMessage('About command selected!');
    else
      inherited; { Default processing }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  MenuH: HMenu;
begin
  MenuH := GetSystemMenu(Handle, False);
  AppendMenu(MenuH, mf_String, cm_About, 'About...');
end;
```

بعد ذلك ، يكرر OnCreate الخطوتين السابقتين ولكن ، هذه المرة ، يقوم البرنامج بتمرير ال Handle من ال application object إلى ال GetSystemMenu . وهذا يعتبر ضرورياً لتعديل قائمة النظام لتطبيق بعيداً عن الرؤية .

## Portable Menus

كثيراً من قوائم التطبيقات لها اوامر متشابهة ، ويمكنك توفير الوقت بتطوير portable menus للمشاركة فيما بين برامج متعددة . يتم استخدام أداتان أساسيتان لإنشاء ال portable menus : ال resources وال templates .

## Menu templates

ان Delphi لديه menu template منشأ متعددة يمكنك إدخالها فى ال PopupMenu objects فى ال MainMenu . لإدخال template :

- ١- أضف واحداً من الـ two objects على الـ form .
- ٢- اضغطه مرتين لفتح الـ Menu Designer .
- ٣- اضغط زر الفأرة الأيمن أو اضغط Alt+F10 لفتح floating pop-up menu .
- ٤- اختر Insert From Template .

٥- اختر موضع قائمتك (قائمة File للتخزين على سبيل المثال) .

بعد إضافة الـ menu template من الصواب ان تعيد تسمية كل الـ item objects للقائمة طبقاً للإرشادات التي ذكرتها . هذا يستغرق لحظات معدودة ، ولكن يساهم بشكل كبير فى الحصول على code يمكن قراءتها . افعل هذا دائماً قبل كتابة عبارات تشير إلى objects باسماءها . ان الاسماء الافتراضية الخاصة بالـ Menu Designer ، مثل Edit1 و Save1 ، لا تعتبر واصفة بالقدر الكافى . ومن الأفضل التسمية بـ FileEdit و FileSave للإشارة إلى أى قائمة ينتمون . ويعتبر FileEditItem ، و FileSaveItem اسماء افضل من السابقة لأنها تذكرك بأنها objects من TMenuItem .

**فكرة:** اذا قم باضافة menu template إلى PopupMenu object ، يقوم Delphi بإنشاء قائمة متداخلة وكأن الـ floating pop-up menu عبارة عن menu bar menu على جانبه . جرب هذا ! هذه التقنية الجيدة لمواجهة التطبيق لا تشغيل فى اغلب الاحيان .

يمكنك ايضاً حفظ أى قائمة على انها template استخدام امر الـ Save as Template الخاص بالـ Menu Designer لتخزين القوائم التى تفضلها مع مخزون الـ template . يخزن الـ Delphi الـ menu templates فى ملف Delphi32.dmt ، الموجود فى دليل \bin الخاص بـ Delphi .

ولا يتم تخزين الـ event handlers وخصائص الاسم مع menu templates لان هذه فى الغالب ستكون مختلفة فى كل تطبيق . و menu templates تعتبر الـ MenuItem objects خالصة . فهى لا تحتوى على ايه code . اذا اردات ان تحفظ الـ code مع menu template قم بإنشاء برنامج

## الباب الخامس : بناء القوائم

اختيارى لكل قائمة ، ثم بعد ادخال ال template فى تطبيق جديد ، قم بنسخ ولصق عبارات من ال module المختبره .

### Menu resource scripts

والطريقة الثانية لإنشاء portable menus هى تصميمها بالطريقة القديمة ، باستخدام resource script command . يمكنك ايضاً استخدام هذه التقنية لإدخال قوائم من تطبيقات اخرى - كتلك التى تم تطويرها فى ال C أو ال C++ ، مثلاً .

على القرص المدمج : يوضح القائمة (٥-٥) ، والقائمة (٥-٦) menu script . وال Multimen.inc يعتبر File include يعرف الثوابت لتمثيل بنود القائمة . ال Multimen.mnu يعتبر resource script يعرف تخطيط فى نص القائمة و Layout . يمكنك العثور عليهما فى الدليل الفرعى Multimen على القرص المدمج المرفق .

#### القائمة (٥-٥) Multimen.inc

```
const
    id_Menu      = 100;
    cm_FileMenu  = 101;
    cm_FileExit  = 102;
```

#### القائمة (٥-٦) Multimen.mnu

```
#include multimen.inc
```

```
id_Menu MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit", cm_FileExit
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo", 201
```

```

MENUITEM "&Redo", 202
POPUP "&Search"
BEGIN
MENUITEM "&Ignore case", 301, CHECKED
POPUP "&Find"
BEGIN
MENUITEM "&Previous", 401
MENUITEM "&Whole words", 402
MENUITEM "&Partial words", 403
END
MENUITEM "&Replace", 303
END
END
END

```

لاستخدام نص الـ Multimen.mnu، أضف MainMenu أو PopupMenu على الـ form. اضغط الـ object مرتين لفتح الـ Menu Designer، واختر امر الـ Menu Designer، واختر امر الـ Insert From Resource. ادخل ملف نص الـ mnu، (بالمناسبة، إنها الطريقة التي أنشأت بها القائمة المتداخلة لتطبيق الـ MultiMen في هذا الباب).

**على القرص المدمج:** بعد إدخال نص قائمة في تطبيق Delphi، يمكنك استخدام الـ Menu Designer لتعديل اوامر قائمتك. ولكن، لا يمكنك بسهولة تعديل النص الأصلي ودمجه مرة أخرى في التطبيق.



ويعتبر محمل نص القائمة الخاص بـ Delphi غير معياري نوعاً ما. يجب ان يكون لكل عناصر القائمة على menu bar بند امر واحد على الأقل. ولكن تطبيقات Windows و Delphi تسمح بالأ يكون لعناصر القائمة أوامراً على الـ menu bar. ان اختيار بند بلا أمر لا يؤدي إلى فتح نافذة قائمة ولكن ينفذ فوراً العنصر على أنه أمر.

لا يسمح الـ Menu Designer بـ command-less menu resources. ولذلك، لا يمكن ان يكون الـ resource script اثنين من اوامر الـ POPUP متتابعين، كما في هذا المثال:

## الباب الخامس : بناء القوائم

POPUP "&Demo"

POPUP "&Help"

وتعتبر القوائم التي ليست لها أوامر قوائم غير معتادة، ولذلك تعتبر هذه المشكلة صغيرة. ولكن، يقدم Borland Pascal 7.0 واغلب نظم تطوير الـ C والـ C++ قوائم بلا أمر، وقد تجد resource script من حين لآخر لا يستطيع Delphi قراءته. يمكنك ابتداء المشكلة بإضافة كلمات BEGIN و END رئيسية لقوائم الـ POPUP. استخدم code مثل:

POPUP "&Demo"

BEGIN

END

POPUP "&Help"

BEGIN

END

## افكار عن الـ Menu Designer:

ان الـ Menu Designer الخاص بـ Delphi له واجهة تطبيق فطرية يسهل التحكم فيها. ولكن، فيما يلي بعض الأفكار التي قد لا تكون واضحة:

- اضغط Ins لإدخال عنصر قائمة خالية أعلى العنصر الذي يتم إبرازه حالياً. قم بإبراز عنصر الـ menu واضغط Ins لإضافة pop-up menu في الـ menu bar.

- اضغط Del لحذف عنصر قائمة. كن على حذر عند ضغط Del - لا يمكنك إبطال حذف قائمة.

- أضف هذه العلامة (-) في الـ Caption الخاص بالقائمة لتكوين خط فاصل. يظهر هذا الخط بالحجم المناسب للـ form وفي وقت التشغيل، ولكن في الـ Menu Designer، يكون الفاصل بطول أمر قائمة طبيعي بحيث يمكنك اختياره وتحريره كأى بند قائمة آخر.

- لإنشاء قائمة متداخلة، قم بإبراز أى عنصر قائمة واضغط Ctrl + السهم المشير الليمين اضغط Esc للعودة إلى المستوى السابق، أو استخدم الفأرة ومفاتيح الأسهم للملاحة عبر القائمة.

● اسحب واسقط عناصر القائمة لإعادة ترتيبها. جرب هذا مع القائمة إنها اسهل طريقة لأداء تغيير كبير على تخطيط قائمة.

## :ActionLists

وواجهة التطبيق الخاصة بالمستخدم توفر طرق عديدة لأداء نفس الوظائف. إنها تزود المستخدمين بمستويات مهارة مختلفة، ويمكن ان يقلل من عبء تعلم كيفية استخدام تطبيق جديد على سبيل المثال، بدلاً من استخدام أوامر القائمة فقط لتحريك برنامج، يمكن لواجهة التطبيق الجيدة ان توفر ازرار و toolbars لتنفيذ نفس الأوامر.

ولكن بالنسبة لواقعي البرامج، كلما زادت المسارات إلى اهداف واجهة التطبيق، زاد العمل الذى يجب القيام به لإنشاء code يمكن الاعتماد عليها. فى برنامج ذى عدة عشرات من الأوامر، فإن إضافة ازرار toolbar تتشارك فى نفس ال code يعود إلى حد كبير للتفاعل بين هذه الأوامر وحالة البرنامج. واحد الامثلة على ذلك هو أنه من الصعب ابطال الأوامر الغير مناسبة بشكل سليم فى الاوقات الصحيحة. وكذلك يصعب تقليل الازدواج فى ال source code - على سبيل المثال، ان كتابة نفس العمليات لـ two command objects مختلفان بطريقتين مختلفتين، والتي قد تؤدي إلى اعطاب فى المستقبل.

يقدم Delphi مفهوم ال action list للمساعدة على حل هذه المشكلات. ان ال ActionList الموجودة فى Standard palette يمكن ان يخزن Action object واحد أو أكثر. ويعد كل object من Action objects نوعاً من الرسالة التى تتعلق بها ال objects الأخرى. لتوفير events وخصائص عامة لإثنين أو أكثر من ال objects الأخرى مثل بنود القائمة والازرار، فعليك ان تبرمج ال Action ليفعل ما تريد به وتلحقه لـ objects الأخرى. وكلاً من هذه ال objects يتخذ سمات Action component. وكأنك برمجت كلاً على حدى.

**ملاحظة:** ان ال ActionList قد تم برمجته فى TActionList class. يعتبر ال ActionList من TActionList class وال ActionList فقط هو الذى يظهر على palette - يتم إنشاء Command objects بواسطة action list editor، والتي يوضح هذا الفصل كيفية استخدامها وهذا الترتيب

Note

## الباب الخامس : بناء القوائم

يشبه الطريقة التي يخزن بها الـ MainMenu component object الطريقة الفردية . MenuItems

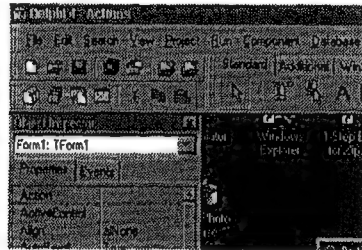
ان مشروع الـ Action فى القائمة (٧-٥) يوضح كيفية برمجة CommandList . يوضح شكل (٥-٥) مظهر وقت التصحيح الخاص بالبرنامج داخل Delphi . ويظهر هذا البرنامج كيف يخزن الـ ActionList الـ Action object والتي تقدم صفات عامة وبرمجة لإثنين أو أكثر من الـ objects الأخرى . لتشغيل البرنامج اتباع هذه الخطوات :

١- افتح مشروع الـ Action باستخدام Delphi (يجب ان يكون لديك Version 4 لهذا العرض) .

٢- قم بتشغيل البرنامج بضغط F9 .

٣- أدخل Quit فى edit box لتشغيل زر Exit . والأمر الذى يحمل نفس الاسم .

٤- يمكنك اختيار إما بند القائمة أو ضغط الزر لإنهاء البرنامج والعودة إلى Delphi .



شكل (٥-٥) : أدخل Quit فى edit box لتشغيل زر Exit  
لخاص بالبرنامج. هذا يظهر كيفية استخدام الـ ActionList

القائمة (٧-٥) : Actions\Main.pas

unit Main;

interface

uses



Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, Menus, ActnList, StdCtrls;

```

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    Edit1: TEdit;
    Button1: TButton;
    Label1: TLabel;
    ActionList1: TActionList;
    Demo1: TMenuItem;
    Exit1: TMenuItem;
    ExitAction: TAction;
    procedure ExitActionExecute(Sender: TObject);
    procedure ExitActionUpdate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.ExitActionExecute(Sender: TObject);
begin
  Close; // Exit program
end;

procedure TForm1.ExitActionUpdate(Sender: TObject);
var
  Flag: Boolean;

```

## الباب الخامس : بناء القوائم

```
begin
// Set Flag True if user types Quit into Edit1
if Lowercase(Edit1.Text) = 'quit'
then Flag := True
else Flag := False;
Button1.Enabled := Flag; // Enable or disable Button1
Exit1.Enabled := Flag; // Enable or disable Menu item
end;
```

على القرص المدمج: لتعليم كيفية استخدام الـ ActionList اتبع هذه الخطوات لإعادة إنشاء تطبيق الـ Action على القرص المدمج في دليل الـ Source\Actions



١- ابدأ تطبيقاً جديداً. أضف MainMenu ، Edit ، Button ، Label ، و ActionList واحداً في كل مرة.

٢- اضغط مرتين على MainMenu1 وقم بإنشاء قائمة Demo مع أمر الـ Exit. وهذا ينشئ الـ Exit1، والذي لا يظهر في الـ form.

٣- حدد خصائص الـ Enabled بـ False للـ Exit1 وللـ Button1 (اختر الـ Exit1 باستخدام قائمة اللائحة الخاصة بالـ Object Inspector). هذا يحدد الخصائص الأولية بـ two command objects، عند تشغيلهما عبر الـ shared ActionList، ينهى البرنامج.

٤- اضغط مرتين على الـ ActionList. مثل الـ MainMenu والـ nonvisual components الأخرى، يظهر الـ ActionList كأيقونة على الـ form، ولكنه غير مرئي في وقت التشغيل. إنك ترى الآن الـ action list editor التي يمكنك ان تنشئ فيها shared Action [انظر شكل (٥-٦)].

٥- اضغط زر الـ New Action (الأول على اليسار) لإنشاء الـ Action. وهذا يسمى Action1 حسب النظام الافتراضي. في هذا العرض، نستخدم Action واحد فقط، ولكن يمكنك إضافة أى عدد من الـ Action تحتجاجة الـ ActionList الواحد.



شكل (٦-٥)، action list editor الخاص بـ Delphi 4  
لوضح هنا، اضغط مرتين لـ ActionList

٦- ان استخدام قائمة لائحة الـ Object Inspector يختار الـ Action1 الذي تم إنشائه في الخطوة السابقة. (وهذا قد تم اختياره في الغالب) سوف تحتاج بطبيعة الحال إلى تغيير اسم هذا الـ object تعكس هدفه. في هذه الحالة، قم بتغيير خاصية الـ Name إلى الـ ExitAction، وحدد خاصية الـ Caption بـ E&xit.

٧- اضغط tab Events الخاص الـ ExitAction في الـ Object Inspector. اضغط مرتين كلاً من OnUpdate و OnExecute، لإنشاء event handler. إملا هذه الـ procedure باستخدام القائمة كمرشد لك- إنني سوف اشرح البرمجة بعد هذه الخطوات.

٨- والخطوة الأخيرة في عملية إنشاء ActionList هي ربط الـ controls التي تشارك في events وخصائص الـ Action. لعمل هذا، اختر الـ Exit1، باستخدام الـ Object Inspector drop-down List، وحدد خاصية الـ Action له بـ ExitAction. بنفس الطريقة، اختر الـ Button1. حدد خاصية الـ Action له ExitAction أيضاً يشارك الآن الزر وبند القائمة في الـ events وخصائص الـ ExitAction. لاحظ ان الـ Captions الخاصة بالـ object المشترك تتغير إلى خاصية الـ Captions التابعة للـ ExitAction.

٩- قم بتشغيل البرنامج بضغط F9. ادخل Quit في edit box لتشغيل الـ command objects، والتي عند اختيارها، تنفذ الـ OnExecute event للـ ExitAction المشترك، وتنتهي البرنامج.

**ملحوظة:** تعتبر خاصية الـ Action، من TAction class، جديدة في Delphi، إنها مقررة كعضو خاص في classes أخرى متنوعة مثل MenuItems، Button، BitBtn، SpeedButton، components أخرى.



لكل Action object مثل الـ ExitAction فى برنامج العرض two events : OnExecute و OnUpdate . ويقدم OnExecute برمجة تنفذ عندما يتم اختيار أى object آخر مثل الـ Button أى الـ MenuItem . وتلك الـ objects الأخرى تحدد خاصية الـ Action الخاصة بها بـ ExitAction المرتبط بالـ code المشتركة ، والتي ، فى هذه الحالة تنهى البرنامج باستدعاء Close .

بطريقة مشابهة ، يقدم الـ OnUpdate Actions الفرصة لتغيير أكثر من الـ objects مشتركة . فى برنامج العرض ، وقمت ببرمجة OnUpdate الخاص بالـ ExitAction لتحديد خصائص الـ Enabled للـ Exit1 والـ Button1 بـ True أو False ، اعتماداً على إذا ما كانت خاصية الـ Text للـ Edit1 تساوى "quit" . يتم استعاء الـ "quit" . يتم استدعاء الـ OnUpdate تكراراً فى أثناء ما يكون (البرنامج غير مشغول ، لذا لا تضيف برمجة مطولة هنا . إن تحديد خصائص الـ True أو False كما حدث فى العرض يعتبر الاستخدام الأمثل لـ OnUpdate الخاص بالـ Action object .

**فكرة:** قم بإضافة ActionList object فى الـ data module (أنظر الباب الثالث ، معلومات حول الـ Form) . هذا يقلل التزاحم فى الـ forms المرئية ، وكذلك يوفر توصلاً سهلاً للـ ActionList والـ Action objects الخاصة به إلى الـ modules الأخرى . ببساطة استخدم الـ data module فى أى form .



### افكار للمستخدم الخبير

● بالرغم من أن الـ MenuItem تعد component ، فلا يمكنك إختيارها جميعاً فى نافذة الـ form لبرمجة خصائص و events مشتركة . لإنشاء event مشترك بين MenuItem objects متعددة ، اكتب procedure واحداً للبند الأول أو أى بند آخر ، وعينة للـ objects الأخرى واحداً تلو الآخر فى نافذة الـ Object Inspector . لا يبدو أن هناك طريقة أسهل لفعل هذا . وبالتبادل ، يمكنك تعيين الـ procedures للـ OnClick الخاص بالـ objects فى وقت التشغيل - ربما فى برنامج الـ OnClick event handlers الخاص بالـ form .

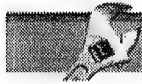
• قم بزيادة مجالات الMenuItem Tag فى onClick event handlers الخاصة بالobject's اطبع هذه القيم قبل أن ينتهى البرنامج . استخدم هذه البيانات لترتيب أفضل لعناصر القائمة- وضع أوامر يتم استخدامها بصورة مكثفة على القمة، مثلاً.

• أدخل نسخة من أمر القائمة الذى تم إختياره على قمة قائمة . وعندما يختار المستخدمون الأوامر ، فإنها تظهر أكثر قرباً من قمة نافذة القائمة . قد تستخدم هذه التقنية لعمل أوامر يتم إختيارها تكراراً ويمكن التوصل إليها- مثل اسماء لل fonts .

• فى الMenu Designer ، قم بتحديد خاصية الMenuItem Break لobject بـ mbBarBreak أو بـ mbBreak لإنشاء horizontally segmented pop-up menus . أنظر القائمة الناتجة فى الform- إنك لا ترى تغييراً مرئياً فى الMenu Designer . وهذه السمات نادراً ما تستخدم فى برمجيات الWindows ، ولكنها متوفرة لإقامة بعض الأنواع من القوائم المعقدة ، وخاصة تلك التى بها أوامر عديدة والتى تجعل نافذة القائمة تمتد بعد الحد السفلى للنافذة .

• استخدام خاصية الAlignment فى الPopupMenu objects لتغيير مكان نافذة القائمة بالنسبة لمؤشر الفأرة . على سبيل المثال ، حدد الAlignment بـ paCenter لوضع قمة النافذة فى مركز مؤشر الفأرة . يعمل الAlignment مع floating pop-up menus فقط المعينة لخاصية الPopupMenu التابعة لل form .

• يوفر الActionList خاصية الImages التى يمكنك استخدامها لتخزين BitBtns لكى تعرض بواسطة visual objects أخرى مثل الMenuItems والBitBtns . أضف الImages الخاصة بك إلى الActionList . لربط كل صورة للAction components الفردية ، استخدم خاصية الImageIndex التابعة لAction object . وهذه الخصائص تجعل من السهل ربط الbitmaps بـ components متعددة .



### المشروعات التى يمكنك تجربتها

١-٥ : اكتب برنامجاً لل standard File menu template . أضف code

من هذا الباب كتنفيذ أوامر الOpen ، الSave ، الSave As ،

## الباب الخامس: بناء القوائم

والأوامر الأخرى. وكذلك أضف code من التطبيق FileMenu لإلحاق اسماء ملفات إلى أسفل القائمة. احفظ الناتج كإحتياطي لبدء مشروعات جديدة متصلة بالملف.

٥-٢: قم بتعديل التطبيق FileMenu للسماح للمستخدمين بتعيين عدد الاسماء المعروضة في قائمة الـ File.

٥-٣: متقدم: اكتب برنامجاً يسمح للمستخدمين ببناء قائمة جديدة في وقت التشغيل. ضمنه أوامراً لإدخال accelerator shortcut keys.

٥-٤: متقدم: باستخدام الـ code الخاص بك من مشروع (٥-٣)، اكتب قوائم بصورة resource script للإدخال في تطبيقات Delphi.

٥-٥: أعد برمجة Menu Designer's templates لاستخدام TMenuItem object أكثر وصفاً. لعمل ذلك، أضف MainMenu object على form ثم اضغطه مرتين، وقم بتحميل كل template في Menu Designer. قم بتعديل اسماء الـ object. بإختيارها وتغيير خصائص الـ Name لها في الـ Object Inspector. على سبيل المثال، يمكنك تغيير الـ Save1 التابع لأمر الـ FileSaveItem. وأخيراً، باستخدام الـ Menu Designer، احفظ الـ template تحت نفس الاسم أو اسماً مختلفاً.

٥-٦: أعد برمجة مشروع الـ Actions. قم بتحويل الـ ActionList object إلى data module (انظر الباب الثالث). قم بتعديل الـ Button objects والـ MenuItem للوصول إلى الـ ExitAction الخاص بالقائمة، على سبيل المثال، على أنه DataModule1.ExitAction.

## ملخص:

- استخدم الـ MainMenu object لإنشاء menu bar نافذة رئيسية.
- استخدم الـ PopupMenu لإنشاء floating pop-up menus تظهر عندما يضغط المستخدمون زر الفأرة الأيمن. يمكنك أيضاً استدعاء الـ Popup method لعرض floating pop-up menus في أى موضع على الشاشة.

• تعتبر جميع العناصر فى ال MainMenu objects وال PopupMenu objects فى TMenuItem class . قم بتعيين قيم لخصائص ال CheckMark ، ال Visible ، وال Enable (وغيرها) فى TMenuItem objects لإنشاء قوائم ديناميكية .

• استخدم خاصية ال Items فى ال MainMenu objects ، ال PopupMenu ، وال MenuItem التى تحتويها كلاً من الأنواع الثلاثة لل menu objects . قم بتحديد قيم فهرس الصف بالمدى من صفر إلى 1 - Count . تعتبر Items من ال TMenuItem ، والتى لها خاصية ال array ؛ وبذلك ، يمكنك استدعاء method ، مثل Insert for Items ، أو يمكنك استخدامها ك array .

• استخدم standard accelerator shortcut keys عند الإمكان . يمكنك تعيين أى مجموعة مفتاح مثل accelerator shortcut . وللحصول على الأداء الأمثل لوظائف ShortCut ، ال ShortCutToKey ، ال TextToShortCut ، وال ShortCutToText الموضحة فى هذا الباب .

• إن أسهل طريقة لإنشاء قوائم ديناميكية هى إضافة ال MainMenu objects متعددة على ال form وتعيينها فى وقت التشغيل لخاصية ال Menu . يمكنك أيضاً تعيين ال PopupMenu objects لخاصية ال PopupMenu الخاصة بال form .

• وهناك أسلوباً آخر سهلاً لإنشاء قوائم ديناميكية وهو تحديد خاصية ال Visible فى ال MenuItem بـ True و False . وهذا يجعل بنود القائمة - أو حتى القوائم بأكملها على menu bar - تظهر وتختفى بموجب تحكم البرامج .

• استخدم ال Add method ، Delete ، Insert ، و Remove لتعديل عناصر القائمة فى وقت التشغيل . على سبيل المثال ، يمكنك استخدام هذه ال procedures لإدخال أسماء ملفات فى أسفل قائمة ال File . إننى أجد من السهل إنشاء بنود قائمة وهمية لهذا الغرض وتحديد خصائص ال Visible لها بـ True عندما يفتح المستخدمون ملفات جديدة .

• يمكنك إنشاء TMenuItem objects جديدة فى وقت التشغيل . استدع ال Create لإنشاء كل object ، واستخدم Add أو Insert لإدخال العنصر فى القائمة . قم بتعيين ال OnClick event handler الخاص بالبند .

## الباب الخامس : بناء القوائم

• أدمج MainMenu objects متعددة بتحديد قيم ال MenuItem GroupIndex وخصائص ال AutoMerge كما هو موضح فى هذا الباب . يمكنك دمج MainMenu objects فى نفس ال form ، أو فى forms مختلفة . إن تطبيقات ال MDI وال OLE تستخدم تقنيات دمج قوائم مختلفة .

• يمكنك إضافة أوامر لقائمة نظام خاصة بتطبيق ما . إن الخطوات المذكورة فى هذا الباب توضح كيفية القيام بهذا ، وتوضح أيضاً كيفية إنشاء ال event handler لرسائل ال Windows .

• استخدم menu templates ، resource scripts لإنشاء portable menus لتشارك فيما بين تطبيقات متعددة . يمكنك أيضاً إنشاء ال menu templates الخاصة بك .

• يضيف Delphi 4 ال ActionList لل Standard . استخدم ال ActionList لإنشاء Action objects تملك ال events . وخصائص يمكن لل components الأخرى مثل ال MenuItem و Button أن تشارك فيها .

إن Delphi يجعل من تحكم الزر فى ال Windows علماً ، كما ستكتشف فى الباب القادم . بالطبع يمكنك إنشاء أزرار ، check boxes ، و radio buttons ، ولكن يمكنك تنسيق واجهة تطبيق المستخدم الخاصة بتطبيقك وذلك باستخدام ال BitBtn component objects الجرافيكى و component objects الأخرى الخاصة بـ Delphi . وسوف نلقى الضوء أيضاً على النص الثابت والتحكم لأعلى وأسفل .





## الباب السادس

# Attaching Buttons and Check Boxes

محتويات هذا الباب:

• Components

• ازرار اساسية

• ازرار ملونة

• إغلاق نافذة

• مجموعات الازرار

• Spin buttons

• Static text

• Up-down button controls

تعتبر الازرار و check boxes هي لب واجهة تطبيق المستخدم الجرافيكية، ويقدم Delphi خزانة أدوات من ال push buttons ، radio buttons ، check boxes ، bitmap buttons ، speed buttons ، organizational components ، button objects ، يمكنك تجميع مجموعات من panels لترتيب نافذة مزدحمة ممتلئة بالازرار والcontrols الأخرى.

في هذا الباب، إنني أوضح كيفية استخدام Delphi's button objects وال components المرتبطة بها مثل SpinEdit ، RadioGroup ، GroupBox ، و

SpinButton. إننى سوف اوضح ايضاً كيفية إنشاء واستخدام bitmap ملونة ،  
والتي تسمى glyphs ، والتي تظهر على ال BitBtn وال SpeedButton .  
يمكنك ايضاً عرض صور ال glyphs ال SpinButton . واخيراً، سوف اشرح  
static text controls (والتي تضيف خصائص مثل ال borders الى standard  
(label text) ، وكذلك الازرار العلوية والسفلية الخاصة بال Win32 ، والتي تشبه  
ال SpinButton ولكنها متصلة فى ال Windows ذو ال ٣٢ بت .

**ملحوظة:** يقدم هذا الباب ال Panel وال SpeedButton . انظر الباب  
التالى للحصول على مزيد من المعلومات حول استخدام هذه  
ال components لإقامة toolbars و status panels .



## : Components

ان ال components المرتبطة بالازرار والخاصة بـ Delphi هي :

● **Bevel** ، ويبدو هذا ال visual component وكأنه فراغات مستطيلة فى  
النافذة. ويمكن لل Bevel ايضاً ان يعرض خطوطاً راسية وافقية والمعروفة جيداً  
لمصممي واجهة التطبيق على انها dips ، speed bumps ، والتي تفيد فى تقسيم  
النافذة الى قطاعات . [Palette: Additional]

● **BitBtn** ، قد تسمى هذه ال component بـ " push button ذو مهارة " .  
يعمل ال BitBtn مثل Button component ولكن يمكنه عرض ايقونة ملونة ؛  
تسمى glyph ، والتي تمثل من الناحية البصرية عمل الزر . [Palette:  
Additional]

● **Button** ، يعتبر Windows button control معيارى محتوى من خلال  
Delphi components . إننى سوف اشرح المزيد عن ال Buttons فى هذا الباب  
ولكنك فى الغالب سوف تريد استخدام ال Button بدلاً منه . [Palette: Standard]

● **CheckBox** ، وهذا ايضاً يعد Windows control معيارى ، ويتكون من  
مجموعة من labels وال boxes يأخذ قيم on و off . إنه مفيد الى درجة كبيرة فى  
تصميم dialog boxes مع مجموعة من خيارات البرنامج . [Palette:  
Standard]

## الباب السادس : Attaching Buttons and Check Boxes

● **GroupBox**، وهو Windows control معياري آخر، وهذا ال component يقوم بتجميع objects- RadioButtons متعددة و ال objects الأخرى . يمكن للمستخدمين ضغط ال Tab للتحرك بين ال GroupBoxes المتعددة، ثم ضغط مفاتيح الأسهم للإختيار من ال controls المتجمعة . وال RadioGroup component، والذي يبسط عملية تطبيق ال RadioButton . [Palette: Standard] .

● **Panel**، هذا ال component يقدم ال platform لتقسيم نافذة مزدحمة لإنشاء toolbars و status panels (سيتم شرحها والباب الثاني) . يمكن ان يظهر على انه سطحاً مرتفعاً، أو يمكن ان يبدو مفرغاً وبه حدود زوايا مائلة متنوعة ان ال Panels لا يكون لها أى input focus، وتعمل كحاوية لل controls . [Palette: Standard] .

● **RadioButton**، يمكنك إضافة ال RadioButton متعددة فى ال Panel أو ال GroupBox، ولكن فى اغلب الحالات، يسهل استخدام ال RadioGroup لإنشاء مجموعات ال RadioButton . ادخل ال strings فى خاصية ال Items لل RadioGroup لإنشاء ازرارك . [Palette: Standard] .

● **SpeedButton**، إنك تستخدم عادة ال SpeedButtons لإنشاء toolbars، كما سيوضح الباب التالى . ولكن، يمكنك إضافة SpeedButtons منفردة على forms . ان تطبيق ال Calc32 الخاص بهذا الباب- آلة حاسبة ذات ٣٢ بت متوفرة على القرص المدمج المرفق بهذا الكتاب- يستخدم ال SpeedButton بهذه الطريقة . [Palette: Additional] .

● **SpinEdit**، موضوع على Samples VCL palette ويعتبر هذا ال component من ال custom component ولكنه مفيد فى حد ذاته . سوف اشرح فى هذا الباب ايضاً كيفية استخدام ال SpinButton المرتبط به : [Palette: Samples] .

● **StaticText**، استخدام هذا ال component الجديد نسبياً لإنشاء static text labels ذات خصائص بصرية مثل sunken border لإنشاء نوافذ قطاعية،

فإن هذا component يعتبر اسهل من تجميع ال Label وال Panel المتعددة .  
[Palette: Additional]

• **UpDown**، هذا component يعتبر 32-bit control والذي يشبه ال SpinButton control وهو موضح أيضاً فى هذا الباب . [ Palette: Win32]

### ازرار اساسية:

يقدم ال Windows ثلاث انواع قياسية من الازرار، والتي تحتويها ال Delphi encapsulates . والازرار الثلاثة وال components المقابلة لها هى :

• **push buttons**، Button component

• **Check boxes**، CheckBox component

• **Radio buttons**، RadioButton component

يتعامل انواع ال components الثلاث مع نفس ال events، على سبيل المثال، يمكنك إدخال عبارات فى ال onClick event handler لأداء اعمال عندما يختار المستخدمون زرأ، radio button، check box، للحصول على تحكم اكبر فى اختيار الزر، استخدم ال OnMouseDown أو ال OnKeyDown . يمكنك أيضاً برمجة اعمال لإطلاق الزر وذلك بإنشاء ال OnMouseUp event .

اجعل قبل أى رمز فى خاصية ال Caption للزر علامة & للإشارة الى Alt-shortcut accelerator key . على سبيل المثال، يمكن للمستخدمين ان يضغطوا Alt+C لاختيار حرفاً فى خاصية ال Caption . على سبيل المثال، يشير ال Caption E&xit الى ال Alt+X على انه button's shortcut key .

### : Push buttons

ان Caption E&xit سهلة الاستخدام . فقط اضغط ال Button object على ال form، وقم بتعيين string لخاصية ال Caption لل button's label . كما رأيت فى الأمثلة العديدة الأخرى، لأداء اعمالاً عندما يضغط المستخدمون زرأ ما، قم بإنشاء event handler ل onClick event الخاص بالزر، وادخل عبارات بين البداية والنهاية .

## الباب السادس : Attaching Buttons and Check Boxes

بالنسبة للزر الافتراضى حسب النظام ، قم بتغيير خاصية ال Default الى True . يجب ان يكون زرأ واحداً على ال form ، (عادة الذى يحمل بطاقة OK) افتراضياً حسب النظام ، والذى يمكن للمستخدمين اختياره ، بضغظ Enter . لربط زر (غالباً يكون Cancel) بمفتاح ال Esc ، حدد خاصية ال Cancel للزر ب True . بغض النظر عن تحديدات ال Cancel وال Default ، يجب عليك إنشاء event handler للأزرار .

وتأخذ الازرار وال controls الأخرى ال Font الخاص بال form . للحصول على عرض افضل ، قم بتغيير ال Font الخاص بال form ليكون TrueType font مثل Arial . ولكن ، عند فعل هذا ، تأكد من اختيار font متاحاً فى كل ال Windows installations . ان ال fonts المثالية المتوفرة فى Windows installation 95 مذكورة هنا . يمكنك اختيار fonts مختلفة للأزرار الفردية وال controls الأخرى .

**ملحوظة:** تعتبر ال TrueType fonts قابلة للتغير من ناحية ، الحجم ، وكذلك bitmap fonts تعتبر قابلة للتغير ، ولكن انظر جيداً فى احجام النقطة المذكورة الخاصة بهم .

Note

• ال TrueType Fonts : Arial ، Arial Bold ، Arial Bold Italic ، Arial Italic ، Courier New ، Courier New Bold ، Courier New Italic ، Bold Italic ، Times New Roman ، Symbol ، Times New Roman Bold ، Times New Roman Bold Italic ، WingDings .

• ال Bitmap Fonts : MS Sans ، Modern ، 15 ، 12 ، Courier 10 ، 8 ، 10 ، 12 ، 14 ، 18 ، 24 ، Roman ، Script ، Small Fonts ، Serif ، 8 ، 10 ، 12 ، 14 ، 18 ، 24 ، Symbol .

فى حالات نادرة قد تريد ان تجبر المستخدمين على اختيار ال push button بعينه ، زر OK مثلاً . لا يمكنك ان تفعل هذا بتحديد خاصية ال Default للزر ب True لان المستخدمين مازال باستطاعتهم ان يضغطوا ال Tab لتحويل ال Focus الى

آخر مثل Cancel . قد يكون هذا محيراً، خاصة للمستخدمين الجدد للحاسوب الذين، لم يكتسبوا خبرة، قد يفشلوا في ملاحظة outline الذي يتركز حول focused control .

وأحد الحلول لهذه المشكلة هو ابطال كل الازرار وال objects الأخرى على form ال بتحديد خصائص ال Enabled بـ False . لفعل هذا قم بتعريف متغير ال I من نوع Integer، واستخدم code كالتالي :

```
for I := 0 to ComponentCount - 1 do
  if Components[I] is TWinControl then
    TWinControl(Components[I]).Enabled := False;
```

وتوصل ال code السابقة الى Components array في ال form لتحديد خصائص ال Enabled لكل ال control objects بـ False . وتختبر عبارة if اذا ما كان TWinControl object (وهي ال class الأم التي تنجدر منها ال control objects مثل الازرار) . اذا كان كذلك، يحدد البرنامج خاصية ال Enabled لل control بـ False بوضع ال Components[I] في ال TWinControl . والوضع هنا أمر ضروري لان محتويات Components array من النوع العام، TComponent، الذي ليس له خاصية Enabled .

بعد تنفيذ ال code السابق، تصبح كل ال controls على ال form، لا يسمح بالتعامل معها . لإنهاء البرمجة، قم بتشغيل زراً واحداً وانقل لل focus الى هذا الزر . يعتبر هذا هو الزر الوحيد الذي يمكن ان يضغطه المستخدمون :

```
OKButton.Enabled := True; { Enable button and set the focus }
OKButton.SetFocus;       { so pressing Enter selects it. }
```

### : Check boxes

تعتبر CheckBox components من ضمن ابسط ادوات تصميم واجهة التطبيق ال Windows ولكن من اكثرها إفادة . قم بإدخال ال CheckBox object في form، وحدد خاصية ال Checked له بـ True لعرض علامة الاختيار صح في المربع، أو بـ False لعدم الحصول على علامة الصح . في وقت التشغيل، يمكنك اكتشاف اذا ما كان ال CheckBox عاملاً أو مغلقاً باستخدام ال code، كما يلي :

## الباب السادس : Attaching Buttons and Check Boxes

```
if MyCheckBox.Checked  
then {do something};
```

يعتبر الـ CheckBox في الغالب on-and-off toggle ولكن يمكن ان يكون ايضاً مفتاح ثلاثي مفتوحاً، أو لا تستطيع التعامل معه أو مغلقاً. ان الـ CheckBox الذي لا يتعامل معه يكون له علامة صح رمادية في مربعه.

لإنشاء CheckBox ثلاثي، حدد خاصية الـ AllowGrayed له بـ True، ثم حدد State بواحدة من ثلاث قيم: cbChecked، أو cbGrayed، أو chUnchecked يمكن للمستخدمين الآن اختيار الـ CheckBox لفتحه أو إغلاقه، أو إيقاف التعامل معه.

إذا كانت الـ AllowGrayed محددة بـ cbGrayed، و State بـ False، تظهر علامة صح لإيقاف التعامل في CheckBox control. يمكن للمستخدمين اختيار الـ control لإغلاقه أو فتحه، لكن لا يمكنهم إعادة CheckBoxes إلى حالته من إيقاف التعامل معه.

**Tip.** فكرة: يمكنك تلوين خلفية الـ CheckBox باختيار قيمة لخاصية الـ Color. على سبيل المثال، اجعل الـ Color بـ clBtnShadow لتلوين الـ CheckBox بنفس لون تحديد النظام لأظلال الزر. أو، اختر لوناً ثابتاً مثل الـ clOlive من قائمة اللائحة لخاصية الـ Color.

لا يمكنك تلوين الـ Button كما تفعل في الـ CheckBox. وهذا يعتبر حد من حدود الـ Windows. للحصول على مزيداً من الازرار الملونة، استخدم الـ BitBtn بدلاً من الـ Button component.

في كثير من الحالات، تقوم بإنشاء CheckBoxes متعددة. إنني أجده من السهل ان تفعل هذا أولاً بإدخال CheckBox object واحد، أضغط Ctrl+C لنسخة في clipboard، ثم أضغط Ctrl+V تكراراً لإضافة objects جديدة على الـ form (قد تختلف هذه المفاتيح اعتماداً على الـ Environment Options الخاصة بك - افتح قائمة Edit للفحص). ثم أقوم باختيار كل objects، غير خصائص الـ Name والـ Caption له، وانقل كل object إلى موضعه النهائي.



## :Radio buttons

وال Windows controls المعيارى الثالث والأخير وهو ال RadioButton ، والذي يحتسبه ال Delphi فى ال RadioButton component. تشبه ال RadioButtons شرائح البطاطس - فواحدة منها لا تكفى. إنك دائماً تستخدم أثنان من ال RadioButton على الأقل (يعتبر ال CheckBox أكثر ملائمة من ال RadioButton للتحكم فى الاغلاق والفتح الفردى). يمكنك تلوين ال RadioButton كما تفعل فى ال CheckBoxes.

عندما تحتوى form على RadioButton متعددة، يمكن للمستخدمين ربما زراً واحداً فقط فى المرة فتحاً أو إغلاقاً. ولكن عندما تدخل مجموعات متعددة من ال RadioButton، فإنها تعمل وكأنها مجموعة فردية واحدة، وهذا ما لا تريد غالباً. والحل هو تجميع كل مجموعة من ال controls بحيث يستطيع المستخدمين اختيار زراً واحداً من كل مجموعة، واضغط Tab لتحويل ال focus من إحدى المجموعات الى الأخرى (والى ال controls اخرى). سوف اشرح المزيد عن تجميع ال control objects فى فصل "مجموعات الازرار"، مؤخراً فى هذا الباب. وكذلك، انظر "استخدام ال Radio Groups" لمعرفة تقنية بديلة لتجميع ال RadioButton.

ال RadioButton تقدم double-click event وهو غير متاح لل RadioButton أو check boxes. اضغط مرتين ال OnDbClick فى ال Object Inspector لإنشاء ال event handler. يمكن للمستخدمين بعد ذلك ان يضغطوا مرة واحدة ال radio button control أو يمكنهم ان يضغطوا مرتين ال object لأداء اعمالاً مختلفة.

### ازرار ملونة:

للحصول على عرض أخذ، استخدم ال BitBtn object بدلاً من ال Windows Button المعيارى. ان ال BitBtn يعمل مثل push button، ولكن يمكنه عرض bitmap ملونة، تسمى glyph، والتي تذكر المستخدمين بل يفعله الزر. ان النظرات قد تكون خادعة، ولكن القليل من التجميل له تأثير بعيد، وال glyphs يمكن ان تساعد على تزيين نافذة تكون رمادية مملة بغير هذه ال glyphs.

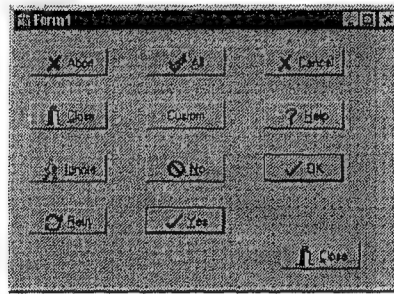
## الباب السادس : Attaching Buttons and Check Boxes

يمكنك أيضاً الاختيار من بين الـ BitBtn المعرفة سابقاً العديد مع الـ glyphs الافتراضية التي تمثل عمليات معيارية الـ Yes ، الـ No ، الـ Close ، الـ Help الـ Retry أو الـ Ignore .

يمكن أيضاً أن يوفر الـ Delphi الـ SpeedButton والذي يشبه الـ glyph bitmap مع SpeedButton وغالباً بلا Caption . وكما سيوضح الباب القادم ، يمكنك استخدام الـ Panels و الـ SpeedButton لإنشاء toolbars ، ولكن يمكنك أيضاً إضافته SpeedButton مستقلة في نافذة الـ form . ويظهر تطبيق هذا الباب Calc32 - آلة حاسبة اعداد صحيحة لوضعي البرامج ذات ٣٢ بت - كيفية استخدام الـ SpeedButton كـ controls فردية .

### ازرار الـ Bitmap

يعتبر الـ BitBtn الأول في Additional VCL palette . ان اسهل طريقة لإستخدام هذا الـ component هي اضافة على الـ form ، ثم تحديد خاصية الـ Kind له بوحدة من القيم الموضحة في شكل (٦-١) على سبيل المثال ، الـ bkHelp . حدد الـ Kind بـ bkCustom لعرض صورة الـ glyph الخاصة بك أو لا تعرض شيئاً . اذا لم تعرض صورة glyph على الـ BitBtn رغم ذلك ، يمكنك استخدام الـ Button .



شكل (٦-١) : form يحتوي على نوع من الـ BitBtn توضيح  
الـ objects القيم المدخلة في خصائص الـ Kind للـ object

**فكرة:** اذا حددت خاصية الـ Kind للـ BitBtn بـ bkClose ، يمكن للمستخدمين اختيار الزر لإغلاق نافذة الـ form الخاصة به . ويقوم الزر بهذا عن طريق التحديد الداخلي للـ ModalResult بقيمة مثل الـ



mrOk . لا يجب عليك كتابة ال code الخاصة بك لتفعل هذا . ولكن ، اذا لم تكن تريد لل Close BitBtn أن يغلق النافذة ، أولاً حدد ال Kind بـ bkClose ثم غيره الى bkCustom . تبقى ال button label وال glyph الخاص به كما هما ، ولكن يكتنك الآن تعيين ال event handler الخاص بك لل OnClick الخاص بال object .

يكتنك تعيين صورة لل BitBtn . اضغط مرتين قيمة خاصية ال Glyph افتح ال Picture Editor . اختر Load وحدد ال bitmap . يكتنك العثور على مجموعة من الملفات فى دليل Images\Buttons الخاص بـ Delphi . بعد تحميل bitmap ، اختر Save لنسخ الصورة فى ملف جديد . اختر OK لنسخ صور من ال bitmaps فى ال BitBtn . وهذه الخطوات تنفذ أيضاً مع ال SpeedButtons .

لا يجب عليك توفير ملف bitmap مع ملف ال code الخاص بتطبيقك . ولكن ، قد تريد نسخ ال bitmap والملفات الأخرى الخاصة بـ Delphi فى أدلة التطوير الخاصة بك حتى تنشئ مجموعة كاملة من ال source files لتطبيقك .

لتعديل موضع ال glyph الخاص بال BitBtn بالنسبة لل caption التابع له ، حدد خاصية ال Layout بـ blGlyphBottom ، أو blGlyphLeft ، أو blGlyphRight ، أو blGlyphTop . قم بتجربة هذا على ال BitBtn لثرى تأثيرها . للحصول على مظهر معيارى . استخدم القيمة الافتراضية blGlyphLeft ، والى تضع ال glyph الى اليسار من ال caption الخاص بالزر . وكذلك قم بتعديل خاصية ال Margin لفصل الصور عن حدود الزر - صفر (أو 0) لعدم الفصل ، 1 لمسافة pixel واحد ، أو two pixels ، وما الى ذلك . لوضع ال glyph فى المركز ، حدد ال Margin بـ -1 .

لمزيد من المعلومات حول glyph bitmaps ، انظر " المزيد عن ال Glyphs " بعد الفصل التالى حول ال SpeedButtons .

### تعديل نمط عرض الـ BitBtn باستخدام خاصية الـ Style

إن النسخ الأولى من Delphi كانت تسمح بتعديل نمط عرض الـ BitBtn باستخدام خاصية الـ Style. للمرجع، لقد ذكرت أوصاف تحديد هذه الخاصية، ولكنها مهجورة الآن، بالرغم من أنه قد يكون لها تأثير بصري في الـ Win32 والـ Windows 3.1 (القيمة الافتراضية للـ BitBtn من (bsAutoDetect)،

• bsAutoDetect: تستخدم نمط الـ bsNew للـ Windows 95 ونمط الـ bsWin31 للـ Windows 3.1.

• bsNew: تستخدم دائماً نمط زر الـ Windows 95 الجديد، مع تأثيرات ظلال أكثر رفعا على الحواف السفلية واليمنى.

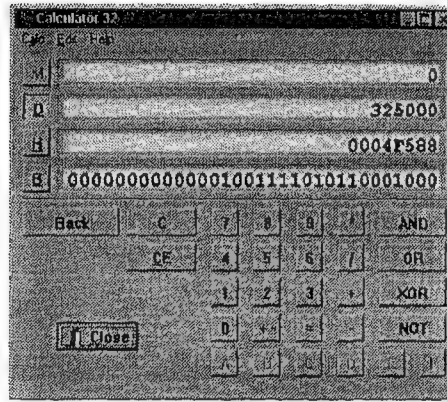
• bsWin31: دائماً يستخدم نمط زر الـ Windows 3.1 مع ظلال أكثر كثافة على الحواف اليمنى والسفلية.

### Speed buttons

تظهر الـ SpeedButton غالباً على toolbar، ولكن يمكنك أيضاً إنشاء SpeedButton منفردة كما سأشرح هنا. يمكن أن تعرض الـ SpeedButton أيضاً أيقونات ملونة تسمى glyphs. أنظر الباب التالي لمزيد من المعلومات عن الـ SpeedButton والـ Panels لإنشاء toolbars.

بشكل عام، إنك تبرمج أحداث الـ SpeedButton كما تفعل في الـ push buttons، ولكن الـ SpeedButtons تتعرف على خمسة events فقط. استخدم الـ OnClick والـ OnDbClick لأداء أعمال للضغط على الفأرة الفردية والمزدوجة. استخدم الـ OnMouseDown و الـ OnMouseMove و الـ OnMouseUp للحصول على تحكم أدق في إختيار الزر- على سبيل المثال، استكشف الـ Shift parameter للـ OnMouseDown ليحدد ما إذا كان هناك زرراً مضغوطاً. (أنظر الباب الرابع لمزيد من المعلومات عن نوع بيانات الـ TShiftState الخاص بهذا الـ parameter).

على القرص المدمج: إن الميزة الأساسية فى ال SpeedButton عن أنواع الأزرار الأخرى هى قدرته على العمل بطريقتين : on/off toggle أو "sticky" button ، ليبقى الى اسفل عندما تدفعه وهذه السمات تجعل ال SpeedButtons الاختيار الامثل فى دليل ال Calc32 . يوضح شكل (٢-٦) ظهور ال Calc32 . ان به SpeedButton objects يمكن ان تعمل on/off toggles (مثل digit entry keys) و "sticky" button تبقى الى اسفل عندما تدفعها (مثل ازرار ال Memory ، ال Decimal ، ال Hexadecimal ، وال Binary فى عنصر قائمة ال Calc .



شكل (٢-٦): لإنشاء آلة حاسبة قمت باستخدام ال SpeedButton التى تعمل ك on/off toggle و "sticky" button وتبقى الى اسفل عندما تضغطها

والسبب الثانى لاستخدام ال SpeedButtons يتعلق بال resource . لأن SpeedButton ليس لها window handles و associated structures فى ال Windows لذا فإن ال form التى بها دسته من ال SpeedButtons تستخدم ذاكرة اقل من دسته من الازرار المعيارية . وتعرض ال SpeedButtons أيضاً بصورة اسرع من ال control المعيارية .

يعتبر ال Calc32 آلة حاسبة للاعداد الصحيحة ذات ٣٢ بت ، مناسبة لمهام البرمجة مثل تشكيل انماط البت والتحويل بين الكسور السداسية ، والثنائية والعشرية يعتبر ال Calc32 بسيط فى الاستخدام اضغط ازرار ال D أو ال H أو ال B لاختيار ادخال عشري ، سداسى أو ثنائى . ادخل قيم باستخدام لوحة المفاتيح أو بضغط

## الباب السادس: Attaching Buttons and Check Boxes

ازرار يظهر دائماً حقول العرض بالقرب من قمة النافذة قيماً في ثلاث جذور بغض النظر عن غط الابدخال. تكون القيم المخزنة في حقل الذاكرة (M) دائماً عشرية. لا يقدم الـ Calc32 قيم من نوع floating-point.

**على القرص المدمج:** ان الـ source code للـ Calc32 طويلة جداً على ان تذكر هنا، بالطبع، جميع الملفات موجودة على القرص المدمج. إننى اشير الى فصول من الـ code من وقت الى آخر. لاختيار التطبيق، افتح ملف مشروع الـ Calc32.dpr فى دليل الـ Calc32.



يوضح الـ Calc32 ان الـ SpeedButtons ليس عليها ان تعرض صور الـ glyph حتى تكون مفيدة. لإنشاء ازرار برنامج، ادخلت خصائص الـ Caption مثل 1، 2، 3، C، CE، وحددت الـ Glyph بـ None (القيمة الافتراضية).

لقد اخترت الـ SpeedButtons للـ Calc32 بدلاً من الـ Button المعيارية لان اثنين أو أكثر من الـ SpeedButtons يمكن ان تعمل مثل مجموعة من الـ RadioButtons. لعمل هذا، قم بتعيين قيم الـ GroupIndex لتحديد مجموعات الـ SpeedButtons اذا كانت الـ GroupIndex تساوى 0 (أى صفر) "وهو القيمة الافتراضية"، يعمل الـ SpeedButtons مثل spring-loaded on/off switch. وتنشئ قيم الـ GroupIndex الموجبة الأخرى مجموعات من الـ SpeedButtons. على سبيل المثال، حدد سلسلة من الـ GroupIndexes الخاصة بالـ SpeedButtons بـ 3. يمكن للمستخدمين ان يختاروا SpeedButtons واحد فقط فى هذه المجموعة، واختيار احد الازرار يغلق تلقائياً الـ control الذى تم اختياره حالياً.

فى الـ Calc32، يكون للـ SpeedButtons الذاكرة (M)، العشرية (D) قيم الـ GroupIndex تساوى 1. ان باختيار احد هذه الازرار يظهر الآخرين. يبقى الزر الذى تم اختياره الى اسفل حتى تختار آخر فى المجموعة.

ان احدى المشكلات التى واجهتنى وانا اكتب الـ Calc32 كانت تعيين event handlers لمجموعات الـ SpeedButtons. لقد اردت ان استخدم اسماء الـ procedure الخاصة بى بدلاً من تلك التى يعينها Delphi حسب النظام الافتراضى للبرنامج. لفعل هذا، قمت بأداء الخطوات التالية:

١- أولاً، اخترت object من المجموعة- على سبيل المثال زر ال A السداسى .

٢- ضغطت مرتين ال object لإنشاء ال ButtonAClick طبقاً لنظام البديل الافتراضى للبرنامج .

٣- بعد ذلك، استخدمت امر ال SearchIReplace التابع ل Delphi لإعادة تسمية كل حدوث لل ButtonAClick بـ DigitButtonClick، والذى يصف بدقة أكثر ما يفعله ال procedure .

٤- وأخيراً، اخترت كل الازرار فى المجموعة، بما فى ذلك الزر الاصلى من الخطوة الأولى . باستخدام tab Events الخاص بال Object Inspector، قمت بتحديد ال OnClick المشترك بـ DigitButtonClick .

### :Glyphs

ال glyph هى Windows bitmap معروضة على شكل SpeedButtons أو BitBtn. بالرغم من انه لا يوجد حدود لحجم ال glyph، فإن الصورة المعيارية ١٦×١٦ pixel، فى ال component، تعتبر خاصية ال Glyph، object من نوع ال TBitmap .

كم هو موضوع فى شكل (٦-٣)، قد تحوى ال glyphs من واحد الى اربع صور منفصلة، مخزنة جنباً الى جنب فى ملف Bitmap، كلاً منهما يجب ان يكون متساوياً فى العرض والطول (غالباً ١٦×١٦ pixel) يعرض Delphi كل صورة لتمثل حالة زر مختلفة:

١- Up (العرض الطبيعى) .

٢- Disabled (معتم؛ Enabled = False) .

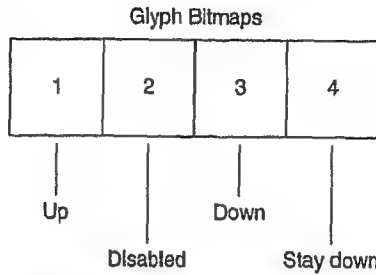
٣- Down (محولة ومن المحتمل معتمه) .

٤- Sty down (لل SpeedButtons فقط) .

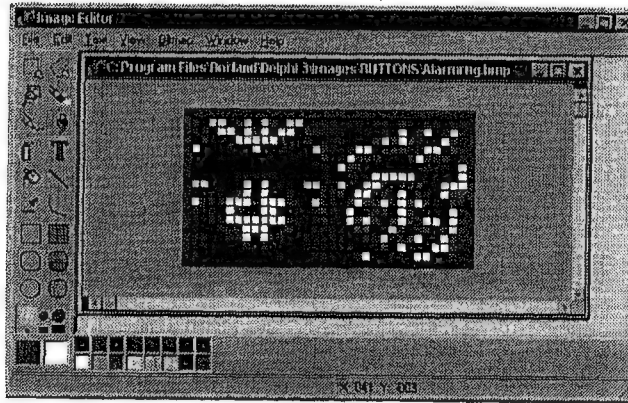
لعرض زر فى حالته الطبيعية، يستخدم Delphi الصورة الأولى . لعرض زر فى حالة دفعه، يعرض Delphi الصورة الثالثة لل glyph. اذا لم يكن هناك صورة ثالثة لل glyph، ينقل Delphi بدلاً من ذلك ال bitmap الى اسفل والى اليمين .

## الباب السادس : Attaching Buttons and Check Boxes

يوضح شكل (٦-٤) المناظر الموسعة والطبيعية للملف Alarmrng.bmp المتاح مع Delphi. لان bitmap يفتقد صور ثالثة ورابعة، ينقل Delphi البت الخاصة بال glyph الى اسفل ويمينا لتمثل حالات الزر المدفوع. يمكنك توفير صور ثالثة ورابعة لإحياء ال BitBtn أو SpeedButton عند اختياره.



شكل (٦-٣)، تحتوي ال Glyph bitmaps من واحد الى اربع صور جنباً الى جنب، كلاً منها له هدف مختلف. على سبيل المثال، يستخدم Delphi الصورة الثانية لعرض زر الابطال



شكل (٦-٤)، ملف ال Alarmrng.bmp يحتوي على glyph من جزئين. يستخدم Delphi الصورة الأولى لحالة الزر الطبيعية وهي انه غير مدفوع، والصورة الثانية للازرار التي تم ابطالها

### تحميل وتعيين glyph bitmaps:

عند تحميل glyph bitmap، حدد خاصية ال NumGlyphs بعدد الصور التي يحتويها. وهذا يجب ان يكون قيمة فيما بين واحد واربعة. في العادي، يعطى Delphi القيمة الصحيحة من حجم ال bitmap. على سبيل المثال، يفترض



Delphi ان ال bitmap الذى له ٣٢ بت عرضاً ١٦×١٦ بت طولاً يحتوى على اثنين من ال glyphs ذات ١٦×١٦ .

يمكنك تعيين صور ال glyphs عندما تصمم form ، أو يمكنك تحميل ملفات bitmap فى وقت التشغيل ربما لإحياء ال BitBtn أو SpeedButton object ، لتغيير مظهره المؤسس على ظروف خارجية . لتحميل ال glyph فى وقت التشغيل ، اصف BitBtn object على ال form ، اضغطاً مرتين ، وادخل هذه العباراتOnClick (قم بتعديل اسم المسار طبقاً للمكان الذى قمت فيه بتركيب ال Delphi):

```
BitBtn1.Glyph.LoadFromFile(
    'c:\delphi\images\buttons\alarm.bmp'); { Or another file }
BitBtn1.NumGlyphs := 2;           { Important step! }
```

تستدعى العبارة الأولى LoadFromFile procedure لل Glyph object لتحميل ملف ال Alarm.bmp المقدم مع Delphi تعدد العبارات الثانية عدد ال glyphs بإثنين.

**ملحوظة:** دائماً حدد ال NumGlyphs بالقيمة السليمة عند تحميل glyph bitmaps فى وقت التشغيل . يحسب Delphi تلقائياً عدد ال glyphs فى ملف bitmap فى وقت التصميم فقط .



يمكنك ايضاً تعيين أى TBitmap object لخاصية ال Glyph لل SpeedButton أو ال BitBtn . ولكن ، بمن المهم ان تفهم ان Delphi يصنع نسخة من ال bitmap لإدخالها فى شكل الزر . إنها مسؤوليتك ان تحذف من الذاكرة أى TBitmap object افتراضية كنت قد أنشأتها . عند تحميل bitmap من ملفات قرص وتعيين الناتجة الى خاصية ال Glyph مباشرة ، لا يجب عليك حذف أى شئ لان الصورة منسوخة مباشرة فى ال object . ولكن افترض انك حددت TBitmap object منفصل ، ربما لأنك تريد ان تستخدمه لأكثر من غرض . أولاً ، قم بتعريف ال TBitmap object ، المشار إليه هنا بـ MyImage :

```
var
    MyImage: TBitmap;
```

## الباب السادس: Attaching Buttons and Check Boxes

قم بإنشاء bitmap object فى الذاكرة باستدعاء الـ Create method الخاص بـ TBitmap class's:

```
MyImage := TBitmap.Create; { Create MyImage object }
```

لديك الآن object فى ذاكرة، يشار إليه بـ MyImage، والذي يمكنك ان تحمل فيه ملف bitmap. يمكنك عندئذ تعيين الـ object لخاصية الـ Glyph الخاصة بالـ BitBtn أو الـ SpeedButton. يمكنك ايضاً استخدام MyImage لأغراض أخرى. ها هي باقى الـ code:

```
MyImage.LoadFromFile  
(c:\delphi\images\buttons\alarm.bmp');  
BitBtn1.Glyph := MyImage;  
{ ... insert other uses for MyImage here }  
MyImage.Free;
```

تستدعى العبارة الأولى الـ LoadFromFile للـ MyImage. تنسخ العبارة الثانية الـ object فى خاصية الـ Glyph للـ BitBtn. أدخل استخدامات أخرى للـ MyImage حيث يشير التعليق. عندما تنتهى من استخدام الـ TBitmap object، حرر الذاكرة الخاصة به باستدعاء الـ free. هذه الخطوة الأخيرة تعتبر أساسية؛ ان نسيت ان تحرر الـ TBitmap الذى انشأته، تبقى اجزاء منه فى الذاكرة حتى بعد انتهاء البرنامج. يقوم تطبيق الـ Delphi بتحرير أية objects يملكها الـ parent object تلقائياً؛ ولكن يجب ان يحدد البرنامج أية structure أخرى مثل الـ bitmap pixels المتعلقة بالـ object. اذا لم يقم البرنامج بتحرير مثل هذه الـ structure، فإن الطريقة الوحيدة لاستعادة الذاكرة المفقودة هى الخروج من التطبيق. وهذا لان الـ Windows ذو الـ ٣٢ بت يخصص resource على اسس سابقة الإنتاج. عندما ينتهى الإنتاج (على سبيل المثال، يخرج المستخدم من التطبيق)، يتم استعادة أى resource ناقصة تلقائياً. ولكن فى الـ Windows ١٦ بت، الطريقة الوحيدة لاستعادة الـ resource مفقودة هى الخروج وإعادة بدء الـ Windows.

ان لون خلفية الـ glyph يساوى لون الـ pixel الواحد فى الركن الايسر السفلى. وكل الـ pixel الأخرى فى الـ glyph التى لها نفس اللون تعتبر شفافة-

بمعنى آخر، عندما يختار المستخدم لون الخلفية الأزرار يحل محل الخلفية للـ glyphs. إذا قام المستخدمون بتلوين الأزرار الخاصة بهم باللون الأحمر الخاص بعربات المطافى، فسوف تكون خلفيات الـ glyphs الخاصة بك مساوية في اللون الأحمر، ولكن على الأقل سوف تظهر كالعائمة على سطح الزر. ولكن للأسف أى pixels حمراء ثابتة سوف تختلط باللون الأحمر الخاص بالزر. والطريقة الأكثر فعالية للتصدي لهذه المشكلة هى استخدام ألواناً مختلفة قدر المستطاع فى صور الـ glyph الخاصة بك. بهذه الطريقة سوف تبدو الـ glyphs الخاصة بك صحيحة حتى للمستخدمين الذين يفضلون تخطيط ألوان الحاسب المكتبى "Hotdog Stand" التابع للـ Windows.

**ملحوظة:** فى العادى، يكون لون (الخلفية) الشفاف للـ glyphs طبقاً للنظام الافتراضى للبرنامج هو لون الـ pixel الأول فى بيانات bitmaps فى الجانب الأيسر السفلى. يمكنك استخدام خصائص الـ TBitmap TransparentColor والـ TransparentMode للتغلب على هذه السمات الافتراضية حسب النظام، على سبيل المثال، قم بتعيين قيمة TColor مثل الـ clAqua للـ TransparentColor. يغير هذا تلقائياً الـ TransparentMode الى الـ tmFixed. للعودة الى استخدام pixel الركن الأيسر السفلى كقيمة اللون الشفاف، حدد الـ TransparentMode مرة أخرى بقيمته الافتراضية، هى الـ tmAuto.

### كيفية تحريك الـ glyph

يمكنك تحريك glyph bitmaps لعرض صور مختلفة عندما يضغط المستخدمون، مثلاً الـ BitBtn. اتبع الخطوات التالية لتجربة هذه التقنية:

١- اضع الـ BitBtn على الـ form لقد استخدمت الاسم الافتراضى حسب النظام هنا وهو الـ BitBtn1.

٢- اختر خاصية الـ Glyph للـ BitBtn1، اضغط الزر البيضوى وقم بتحميل ملف Dooropen.bmp من دليل الـ Images\Buttons الخاص بـ Delphi.

## الباب السادس: Attaching Buttons and Check Boxes

٣- اضعف التعريف التالى فى public section لل TForm1 class (يمكن وضعهم فى ال private section ايضا):

```
DoorShutBmp: TBitmap;
```

```
DoorOpenBmp: TBitmap;
```

٤- قم بإنشاء ال bitmap objects وقم بتحميل الصور فيها بإضافة هذه العبارات الى ال OnCreate event الخاص بال from (انسخ ملفى ال bitmaps، فى الدليل الحالى من دليل Images\Buttons الخاص ب Delphi):

```
DoorShutBmp := TBitmap.Create;
```

```
DoorOpenBmp := TBitmap.Create;
```

```
DoorShutBmp.LoadFromFile('doorshut.bmp');
```

```
DoorOpenBmp.LoadFromFile('dooropen.bmp');
```

٥- حرر الصور بإضافة، هذه العبارات الى ال OnDestroy الخاص بال form:

```
DoorShutBmp.Free;
```

```
DoorOpenBmp.Free;
```

٦- قم بإنشاء ال OnMouseDown event الخاص بال BitBtn1 (تأكد من اختيار الزر وليس ال form. اضعف هذه العبارة الى ال procedure:

```
BitBtn1.Glyph := DoorShutBmp;
```

٧- قم بإنشاء ال OnMouseUp event الخاص بال BitBtn1، واضف هذه العبارة الى ال procedure:

```
BitBtn1.Glyph := DoorOpenBmp;
```

٨- اضغط F9 لتشغيل البرنامج. عندما تضغط ال BitBtn، يغلق الباب. عندما تطلق زر الفأرة يفتح الباب.

وطريقة أخرى بتحريك ال glyph هى بإنشاء bitmap واحدة تحتوى على صور منفصلة لحالات الزر العليا السفلى. على سبيل المثال، يمكنك جمع ملفات ال bitmaps : Dooropen.bmp و Doorshut.bmp، وتعيين ال glyph

المكون من اربعة اجزاء لـ BitBtn . بهذه الطريقة تزيل الحاجة الى تحميل وتحرير bitmap objects المفردة فى وقت التشغيل .

### مجموعات الازرار:

فى النوافذ المزدحمة- configuration dialog على سبيل المثال- يعتبر من الصواب تصنيف الازرار المتعددة فى مجموعات . قم بترتيب مجموعات الازرار بثلاث طرق عامة :

- استخدام ال Bevel وال Panel لإنشاء sections فى ال form تظهر ذات فراغ أو مرفوعة . ادخل ازراراً فى هذه ال objects .

- ادخل ازراراً فى ال GroupBox .

- قم بإنشاء RadioGroup ، والذي يمكنه تلقائياً توليد RadioButtons متعددة . ببساطة أضف button labels فى Items array الخاص بال RadioGroup (المزيد عن هذا فيما بعد) .

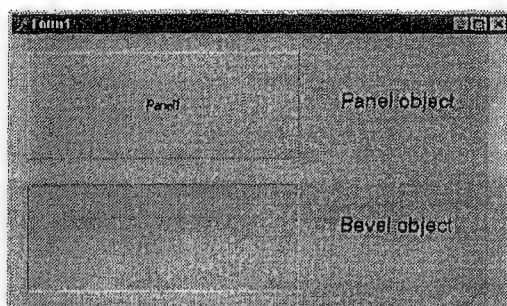
### استخدام ال Bevel وال Panel :

ال Bevel وال Panel ، تساعدك فى إنتاج عرض جذاب . ان ال Bevel موجود على ال Additional VCL ، وال Panel على ال Standard . إسقط هذه ال objects على ال form لتجربتها .

يمكن ان تستجيب Panel objects لـ events مثل ال OnClick وال OnMouseDown . وتشبه ال Panels الازرار البسيطة التى لا تظهر مدفوعة عندما تغطها . ال Bevel تشبه ال Panel ، ولكن لا تعرف على أية events . وتعتبر ال Panels مفيدة أيضاً فى إنشاء toolbars و status panels كما سيوضح الباب القادم . وال Panels لها window handles ، ولذلك ، فإنها تستخدم system resources أكثر من ال Panels . ولكن ، يمكن للـ Panels ان تحتوى controls اخرى مثل ال Bevels ، وتعتبر ال Bevels بصرية ، ولا لها تفتقد ال window handles ، لا يمكن للـ Bevels ان تحتوى على Buttons أو controls أخرى .

شكل (٥-٦) يوضح ال form الافتراضية مع ال Panel وال Bevel . للـ Panel تعليق افتراضى (وهو Panel فى الشكل) ، ولكن يمكنك حذف قيمة خاصة ال Caption لعرض سطح خالى .

## الباب السادس: Attaching Buttons and Check Boxes



شكل (٦-٥): form افتراضية مع Panel objects و Bevel،  
والتي تفيد في ترتيب عرض مزدحم الى مقاطع بصرية

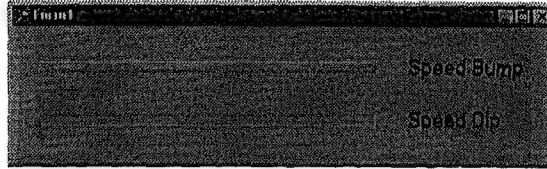
قم بتعديل خصائص الـ BevelOuter، الـ BevelWidth، الـ BorderStyle، والـ BorderWidth لمظاهر الـ Panel المختلفة. قم بتعديل خاصية الـ Shape لتغيير مظاهر الـ Bevel. (وقد يبدو محيراً أن الـ Panels تستخدم خصائص الـ Bevelxxx لتحديد مظاهرها، ولكن هذه تسمية ليس لها مدلول. ان الـ Panels و الـ Bevels انواع مختلفة من الـ components).

لا يجب أن تتخذ الـ Bevel اشكال مربعات. يمكنك أيضاً استخدامها لإنشاء خطوط مقسمة، التي يسميها بعض واضعي البرامج speed dips و speed bumps. على سبيل المثال حدد خاصية الـ Shape الـ Bevel بـ bsBottomLine لعرض speed dip أفقي، والذي يفيد في إنحناء نافذة لتكون قطاعات علوية وسفلية. استخدم bsLeftLine أو bsRightLine لإنشاء حواجز رأسية. قم بتعيين الـ Style ليصبح bsRaised لإنشاء speed bump. يوضح الشكل (٦-٦) نماذج لـ speed dips و speed bumps لكي اجعل هذا. مرئياً بشكل واضح هنا، فقد حددت. خصائص الـ Shape لأثنان من الـ Bevel بـ bsBox؛ ولكن يمكنك ان تحدد بـ bsBottomLine، أو bsFrame، أو bsLeftLine، أو بـ bsRightLine، أو bsTopLine لإنشاء speed dips و speed bumps على خط واحد.

### استخدام الـ GroupBoxes

الـ GroupBoxes، يحتويه الـ Delphi في component. استخدم الـ GroupBoxes لإنشاء مجموعة من الـ check boxes، radio buttons،

وother controls أخرى. يمكن للمستخدمين ان يضغطوا Tab لينتقلوا من أحد. ال GroupBoxes الى آخر، ويمكنهم ضغط مفاتيح السهم وقضيب المسافة للاختيار من بين ال controls المجتمعة.



شكل (٦-٦): form حسب النظام الافتراضى وبها speed dip و speed bump

وتقيّد ال GroupBoxes الى حد كبير وترتيب radio button فى مجموعة منطقية. على سبيل المثال، يمكنك إضافة radio button ومتعددة فى كل واحد من الأثنان من ال GroupBox. يمكن للمستخدمين عندئذ اختيار أحد الأزرار من أى من المجموعتين اذا لم تضاف ال radio فى ال GroupBoxes، يمكن للمستخدمين، ان يختاروا أحد الأزرار فقط فى النافذة.

يوضح الشكل (٦-٧) form بها إثنين من ال GroupBoxes، فى كل منهما ثلاث أزرار.

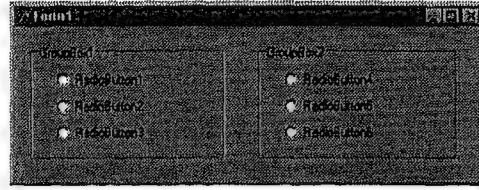
لإنشاء هذه النافذة:

١- أضف أثنان من ال GroupBox الاثنين أولاً.

٢- أضف RadioButtons الى ال GroupBox objects.

٣- ان ضغط مؤشر الفأرة داخل ال GroupBox لإدخال ال RadioButtons (أو أى control آخر) يخبر Delphi ان يربط ال object الذى تم إدخاله بال GroupBox.

حتى يعمل عملية ال tab بشكل صحيح، يجب اختيار RadioButton واحد على الأقل من كل مجموعة. حدد خاصية ال Checked بـ True لـ RadioButton واحد فى كل GroupBox. وهذا من حدود ال Windows على radio button controls المجموعة.



شكل (٦-٧): استخدام الـ GroupBoxes  
لترتيب RadioButtons في مجموعة منطقية

إذا أضفت RadioButtons قبل ان تنشئ الـ GroupBoxes الخاصة بهم، لن تعمل الازرار كمجموعات متميزة. ان سحب الـ RadioButtons في الـ GroupBoxes لن يكون مفيداً - تبقى objects مرتبطة بال components التي تم إسقاطها عليها في أول الأمر (ال form أو control يحتوية آخر). اذا واجهت هذه المشكلة، اختر الـ RadioButtons، وأحفظهم في الـ clipboard اختر الـ intended GroupBox، والصق الـ RadioButtons من الـ clipboard إلى الـ container الجديدة يمكنك أيضاً حذف الازرار وإعادة إدخالها من على VCL palette مباشرة إلى الـ GroupBox، ولكن هنا تفقد اية تغييرات في الخاصية تكون قد صنعتها للـ controls.

لجمع radiobuttons أو control objects آخر في وقت التشغيل، حدد خاصية الـ Parent بالـ GroupBox للزر المختار. (يمكنك ان تفعل هذا في وقت التشغيل فقط لان خاصية الـ Parent لا تكون متاحة في نافذة الـ Object Inspector). على سبيل المثال، لربط ثلاثة RadioButton objects بالـ GroupBox1، استخدم عبارة مثل التالية، ربما في الـ form's OnCreate:

```
RadioButton1.Parent := GroupBox1;  
RadioButton2.Parent := GroupBox1;  
RadioButton3.Parent := GroupBox1;  
RadioButton1.Checked := True; { Select one grouped button }
```

والطريقة الصائبة لتعرف اذا كانت الـ RadioButtons قد تم تجميعها بشكل سليم هي ان تسحب الـ GroupBox. اذا كانت الـ RadioButtons مثل البط الصغير الذي يتبع أمه، فيكون قد أتم بهذه المجموعة. اذا لم يكن كذلك، ففي الغالب يكون الازرار قد تم ربطها بال form. قص والصق الازرار لحل المشكلة.



يمكنك أيضاً تجميع الـ GroupBox، ولكن لأن هذه الـ controls تعمل منفردة، فلا يوجد غالباً سبب وجيه لكى تجمعها فى مجموعة. فى غالبية الحالات، يكن من الجيد استخدام الـ Bevels أو الـ Panels كما هو الحال فى ترتيب مجموعات بصرية من الـ GroupBox.

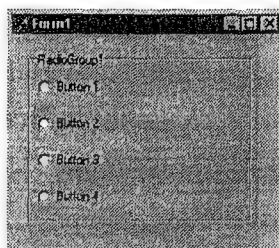
لكى تجعل مجموعات RadioButtons تعمل كما تريدها عندما يضغط المستخدمون مفتاح الـ Tab، فعليك ان تعبث فى نظام الـ Tab الخاص بالـ form بعد إنشاء الـ GroupBox وبإضافة الـ RadioButton، اختر الـ form واختر Edit/Tab Order لتحديد الترتيب الذى بموجبه يستطيع المستخدمون ضغط الـ Tab للتنقل بين المجموعات. بعد ذلك، اختر كل GroupBox، ومرة أخرى استخدام الـ Edit/Tab Order، ولكن هذه المرة، للتأثير على ترتيب الـ Tab RadioButtons المجمعة.

### استخدام الـ RadioGroups

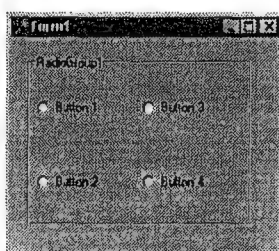
يدرك غالبية المطورين ان الـ GroupBoxes والـ RadioButtons لها طبيعة مشاكسة كالثيران فى الحلقة. والطريقة الاقل جهداً لإنشاء مجموعات RadioButtons، أضف الـ RadioGroup component على الـ form. لا تدخل الـ RadioButtons فى الـ RadioGroups. بدلاً من ذلك، اختر خاصية الـ Items للـ RadioGroups، واضغط قيمتها مرتين أو اضغط الزر البضاوى لفتح الـ String list editor فى الـ Delphi.

اضف الـ labels لكل الـ radio button بالترتيب الذى تريده ان تظهر فيه. هذا هو كل ما عليك ان تفعله لإنشاء مجموعات من الـ RadioButton فى الـ RadioGroup.

واختيارياً حدد خاصية الـ Columns للـ RadioGroup بعدد الاعمدة لعرض الـ RadioButtons. يوضح الشكل (٦-٨) اربعة RadioButton فى الـ RadioGroup مع الـ Columns مساوية الى (1). يوضح شكل (٦-٩) نفس الـ RadioGroup مع الـ Columns محددة بـ 2.



شكل (٨-٦): اربعة RadioButton  
في RadioGroup وال Columns محددة (1)



شكل (٩-٦): نفس الازرار من  
شكل (٨-٦) مع ال Columns مساوية لـ (2)

استخدم خاصية ال `ItemIndex` لفحص الزر الذي تم اختياره حالياً. حدد ال `ItemIndex` بـ (-1) "وهي القيمة الافتراضية) لإبطال اختيار جميع الازرار في مجموعة ما. حدده بـ (0) لاختيار الزر الأول، وبـ 1 لاختيار الزر الثاني، وما الى ذلك.

يمكنك إنشاء radio buttons في RadioGroup في وقت التشغيل. على سبيل المثال، استخدم هذا المتغير وال procedure للسماح للمستخدمين بإضافة radio buttons جديدة في مجموعة:

```
var
  S: String;
begin
  if InputQuery('Input', 'Enter Radio Button', S) then
    RadioGroup1.Items.Add(S);
end;
```

ال Items array عبارة عن string list من نوع ال TStrings. إستدع Add method الخاص بالقائمة لإدخال string جديد، والذي يجعل ايضاً ال RadioButton ينشئ RadioButton جديد مستخدماً ال strings ك label. وتساوى خاصية ال Items Count عدد strings فى خاصية ال strings، ولذلك، فهى ايضاً تساوى عدد ال RadioButtons الذى تملكه ال RadioButton. وتساوى ال ItemIndex، ال button's index الذى اختياره حالياً.

استخدم هذه القيم للتكرار من خلال ال button labels، كما فى العبارة التالية، التى تستدعى ال AnyProcedure (غير موضح):

```
for I := 0 to RadioGroup1.Items.Count _ 1 do
  AnyProcedure(RadioGroup1.Items.Strings[I]);
```

أو، استخدم عبارة ال with لجعل ال code السابقة أكثر وضوحاً، ولتقليل الإشارة الى ال RadioGroup1 من اشارتين الى واحدة:

```
with RadioGroup1, Items do
  for I := 0 to Count - 1 do
    AnyProcedure(Strings[I]);
```

لاحظ ال syntax الخاصة لعبارة ال with، باستخدام الفصلات لفصل أكثر من identifiers ان السطر الأول فى المثال السابق مساو لـ:

```
with RadioGroup1 do with Items do
```

استخدم ال ItemIndex لتحديد الزر الذى تم اختياره. فمثلاً، عبارة ال if يمكن ان تقوم بعمل لزر معين معرف بواسطة ال label الخاص به:

```
with RadioGroup1, Items do
  if Strings[ItemIndex] = 'Button2' then...
```

حدد دائماً زراً واحداً بحالة فتحه لكل ال RadioGroup؛ والا، لا يمكن للمستخدمين ان يضغطوا ال Tab لتحويل ال focus الى المجموعة (إحدى غرائب ال Windows). إما ان تحدد ال ItemIndex بـ (0) عندما تصمم ال RadioButton، أو تدخل ال code التالية فى ال OnCreate الخاص بال form:

## الباب السادس: Attaching Buttons and Check Boxes

```
with RadioGroup1, Items do
```

```
if Count > 0 then
```

```
  ItemIndex := 0
```

```
else
```

```
  ItemIndex := -1;
```

يمكنك أيضاً استخدام هذا الـ code للـ RadioGroup الذى ينشئه البرنامج فى وقت التشغيل ؛ فى مثل هذه الحالة يجب ان تسمح بإمكانية ان تكون المجموعة بلا ازرار . فى هذه الحالة ، يجب ان تحدد الـ ItemIndex بـ -1 كما هو موضح هنا .  
تحتوى الـ RadioGroup على قائمتين تمثل الـ RadioButtons المجمعة .  
يحتوى Items arrays على string list للـ button labels وهى متاحة فى وقت التصميم ووقت التشغيل و Components array ، المتوفر فى وقت التشغيل فقط ، يحتوى على إرشادات عن الـ RadioGroup objects الحقيقية .

افحص الـ Components array والعناصر المتعلقة به ، اصف  
RadioGroup object على الـ form ، اصف . button labels متعددة فى  
Item's string list . اصف Button وقم بإنشاءOnClick مع هذه العبارات :  
with RadioGroup1 do  
if ItemIndex >= 0 then  
 ShowMessage('You selected ' + Items.Strings[ItemIndex]);

قم بتشغيل البرنامج واضغط الزر لعرض الـ radio button's label التى تم اختيارها حالياً . اترك البرنامج وارجع الى Delphi ، ثم حدد Breakpoint على عبارة الـ with . لفعل هذا ، اضغط مرة واحدة مؤشر الفأرة فى أقصى يسار السطر ، أو حرك مؤشر لوحة المفاتيح الـ wامض الى السطر واختر Run/Add Breakpoint .  
فى أى من الطريقتين ، يلون Delphi العلامة العلوية للعبارة باللون الاحمر ، والتى تشير الى ان البرنامج سوف يتوقف قبل ان تنفذ هذه العبارة مباشرة .

قم بتشغيل البرنامج مرة اخرى واضغط الزر . يتوقف البرنامج عند breakpoint . حرك المؤشر الـ wامض فى أى مكان داخل الـ RadioButton1 ، واضغط Ctrl+F7 لفتح Evaluate/Modify dialog الخاص بـ debugger .  
يجب ان ترى الـ RadioGroup1 فى مجال الـ Expression وقائمة يقيم

الخصائص فى مربع Result. اذا لم يكن كذلك ، أدخل RadioGroup1 فى حقل ال Expression واضغط Evaluate. وتعتبر هذه تقنية مفيدة تضع قائمة باعضاء ال objects الغير متوفره بشكل عام من خلال نافذة ال Object Inspector.

وتعتبر القوائم المطولة عن القيم فى ال objects وال records صعبة فى فك شفرتها فى ال Evaluate/Modify dialog. لعرض كل خاصية بإسمها وقيمتها ، أدخل "R" (لا تدخل علامات التنصيص ، ولكن لا تهمل الفصلة قبل ال R) بعد identifier فى حقل ال Expression. على سبيل المثال ، اذا كنت متبعاً ، اكتب تعبير R, RadioGroup1 واضغط Enter.

**Tip** فكرة: قم بتشغيل tool-tip expression evaluation باستخدام ال Code Insights page tab و Tools/Environment Options. يمكنك عندئذ نقل المؤشر الى اسم object مثل RadioGroup1 ، توقف قليلاً ، وانظر ان قيمة ال object معروضة فى pop-up window صغيرة.

يجب ان ترى الآن القيم التالية ، بالإضافة الى اخرى كثيرة ، فى ال Evaluate/Modify dialog. اضغط Evaluate اذا لم ترى ذلك. يتبع اسم الخاصية ال class method المتعلقة بها. فمثلاً ، ها هنا أول سطرين من حقل ال Result:

Components:<GetComponent>;

ComponentCount:<GetComponentCount>;

يحتوى ال Components على ال RadioButton objects التى تنتمى الى ال RadioGroup. تساوى ال ComponentCount عدد هذه ال objects. استخدم هاتين القيمتين للتوصيل الى ال RadioButtons منفردة فى ال RadioGroup. على سبيل المثال ، لتحديد خاصية ال Checked بـ False لكل ال RadioButtons فى مجموعة ، يمكنك استخدام for loop مثل هذه :

with RadioGroup1 do

for I := 0 to ComponentCount - 1 do

TRadioButton(Components[I]).Checked := False;

## الباب السادس : Attaching Buttons and Check Boxes

أو ، لاختيار معناها ، استخدم هذه العبارة :

with RadioGroup1 do

TRadioButton(Components[2]).Checked := True;

عند تذكر الـ Components array ، يجب ان تخبر الـ compiler أى نوع من الـ objects يحتوى عليه الـ array . افعل هذا باستخدام اسم الـ class (هنا TRadioButton ) وكأنه function تصف object من نوع محدد . قم بتمرير indexed Components reference لإقامة هذا النوع . يمكنك عندئذ التوصل الى خصائص object مثل Checked .

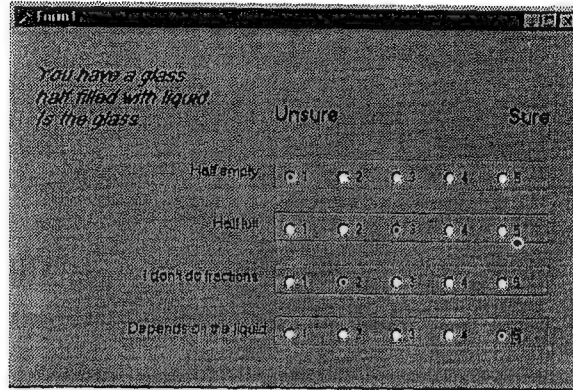
إذا كنت تتابع ، إرجع الى Delphi ، واضغط F9 لاستمرار عمل البرنامج ، الذى توقف عند الـ breakpoint أو وقف عمل البرنامج للعودة الى ثط التصميم فى Delphi . يمكنك عندئذ إزالة الـ breakpoint ، أو بدء مشروع جديد .

وتفيد الـ RadioGroups أيضاً فى تصميم form استطلاع الرأى ، option form ، multiple ، choice form . قم بإنشاء RadioGroup متعددة ، وادخل مسافات خالية ، علامات ترقيم ، خانات رقمية ، أو رموز أخرى فى Items array فى هذه الحالة ، تكون فى الغالب لا تريد labels كاملة- إنك تريد ان تعرض فقط دوائر الـ RadioButton control . حدد خاصية الـ Columns مساوية لعدد الازرار ، قلل عدد الـ RadioGroup قدر ما تستطيع . يوضح شكل (٦-١٠) مثال افتراضى يستخدم الـ RadioGroup لجانب سيكولوجى استوحيته من صديق طبيب نفسى الذى يقوم باختبارات مشابهة .

بعض النسخ من وثائق Delphi تقول انه بإمكانك وضع انواع أخرى من الـ controls (بغلاف radio buttons) فى الـ TRadioGroup هذا غير صحيح . يمكنك إضافة RadioGroup فقط فى الـ RadioGroup .

### انواع أخرى من الـ RadioGroup :

قد تعرف انه بإمكانك إضافة الـ RadioButton مباشرة فى نافذة الـ form . إذا كنت تريد مجموعة واحدة فقط من radio buttons ، فلا يجب عليك إدخالها فى GroupBox ولا تحتاج أن تستخدم RadioGroup . فقط أدخل ما تستطيع إدخاله من RadioButton على قدر حاجتك فى أى form .



شكل (٦-١٠)، تساعد الـ RadioGroup فى إنشاء اختبارات

متعددة الاختيارات واستطلاعات رأى كما فى هذا الجانب الافتراضى

ومن غير المعروف، (ولكنه ليس سرّاً)، أنه بإمكانك أيضاً تجميع الـ RadioButton المتعددة فى Panel. قم بتجربة هذا:

١- أضيف Panels على الـ form، ثم أضيف الـ RadioButtons المتعددة فى كل panel.

٢- حدد خاصية الـ Checked لأحد الـ RadioButton فى كل مجموعة بـ True.

٣- قم بتشغيل البرنامج.

كما توضح هذه التجربة، لا يوجد اختلافات عملية بين الـ GroupBox والـ Panel - على الأقل من ناحية قدرتها على تجميع الـ radiobuttons المتعددة. ولكن يظهر تعليق الـ GroupBox على حده العلوى - يعرض الـ Panel تعليقه فى الداخل، وقد تحتاج أن تحذفه لتفسح مكاناً للـ radiobuttons والـ controls الأخرى.

**على القرص المدمج:** إحدى الميزات فى استخدام Panels كمربعات

مجموعة هى إمكانية الجرافيك 3D الخاص بالـ Panel، والتي يفتقدها الـ

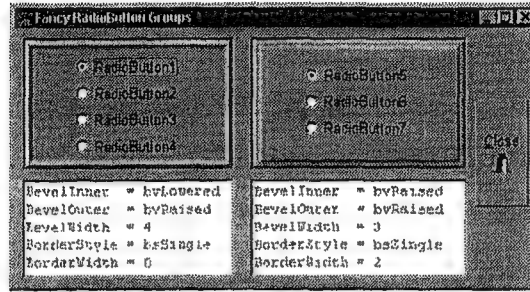
GroupBox. كما يوضح شكل (٦-١١)، يمكنك استخدام الـ Panels

لإنشاء مجموعات RadioButton خيالية. والنص الموجود تحت كل مجموعة من الـ RadioButtons يوضح قيم الخاصية التى تنتج تأثيرات الـ 3D الموضحة.



## الباب السادس : Attaching Buttons and Check Boxes

وتطبيق ال Fancy الذى ليس له code هام ، موجود على دليل ال Source\Fancy فى القرص المدمج المرفق بهذا الكتاب .



شكل (٦-١١): التطبيق Fancy يظهر كيفية

استخدام ال Panel لإنشاء مجموعات 3D RadioButton

من الممكن إضافة objects من انواع مختلفة فى ال GroupBox وال Panel . على سبيل المثال ، يمكن ان يحتوى ال Panel على ثلاثة من ال RadioButtons واثنين من ال CheckBoxes . يمكن للمستخدمين ان يضغطوا Tab للتنقل بين المجموعات . والنتيجة النهائية قد يكون مفيدة فى البرامج التى تحركها لوحة المفاتيح الى حد كبير . ولكن لان المستخدمين يختارون ال CheckBoxes منفردة ، فإن إضافتها لل Panel أو ال CheckBoxes ليس له ميزة حقيقية .

## : Spin Buttons

تجد ال Samples page tab على VCL palette ، button controls إضافيتين ، وهما ال SpinButton وال SpinEdit . قم بتعيين ال glyphs الخاصة بك لخصائص ال DownGlyph فى ال UpGlyph التابعة لهذه ال objects لعرض اسهم مخصصة مشيرة الى أعلى أو أسفل أو لعرض ال objects أخرى . وال components هما :

• **SpinButton**: مزدوج ذو أزرار علوية وسفلية . إن ضغط أحد الأزرار يستدعى ال OnUpClick أو ال OnDownClick .

• **SpinEdit**: هو ال SpinButton ملحق بـ edit field إن ضغط السهم العلوى أو السفلى للزر يزيد أو ينقص قيمة صحيحة فى ال edit field . يمكن للمستخدمين أن



يدخلوا قيم فى هذا الحقل ، والذي يتعرف ايضاً على اوامر ال cut ، والنسخ ، واللصق .

### استخدام SpinButton components :

لإستخدام SpinButton components إتبع هذه الخطوات :

١- أضف ال SpinButton على ال form واضف ال code فى OnUpClick و OnDownClick الخاصة به . اضف ال Label على ال form وقم بتعريف المتغير التالى فى قطاع ال TForm1 class's private :

```
private
```

```
Count: Integer;
```

٢- قم بإنشاء ال OnCreate الخاص بال form وابدأ ال Count بصفر :  
zero.

```
Count := 0;
```

٣- قم بإنشاء ال OnUpClick و ال OnDownClick الخاصة بال SpinButton's . استخدم البرمجة الموجودة فى القائمة (٦-١) لزيادة أو نقص ال Count ، ولعرض قيمة ال Count فى ال label .

```
القائمة (٦-١) : استخدم ال SpinButton وهذه البرمجة فى ال OnUpClick وال
OnDownClick لزيادة وخفض متغير ال Count وعرضه فى ال label
procedure TForm1.SpinButton1DownClick(Sender: TObject);
begin
    Dec(Count);
    Label1.Caption := IntToStr(Count);
end;
```

```
procedure TForm1.SpinButton1UpClick(Sender: TObject);
begin
    Inc(Count);
    Label1.Caption := IntToStr(Count);
end;
```

## الباب السادس : Attaching Buttons and Check Boxes

استخدم خصائص الـ UpGlyph والـ DownGlyph في الـ SpinButton لعرض glyphs للأزرار العلوية والسفلية الخاصة بالـ object إما أن تختار من مخزن Delphi، bitmaps للأسهم، أو تنشئ glyphs خاصة بك.

قد لا تحتوي glyphs الـ SpinButton على صور متعددة، ولذلك، فإن استخدام bitmaps المقدمة من Delphi (والتي تحتوي جميعها على صور مزدوجة)، يجب عليك أن تحولها إلى glyphs أحادية الصورة افعل هذا بنسخ الملفات التي تريدها على دليل آخر - ملفات الـ Arrow1d.bmp والـ Arrow1u.bmp مثلاً من دليل Images\Buttons الخاص بـ Delphi. افتح هذه الملفات في الـ Image Editor من قائمة الـ Tools. لكل ملف، اختر الـ ImageAttributes، قم بتغيير الـ Width من ٣٢ إلى ١٦ (أو اجعله مساوياً للـ Height)، وأبطل إختيار صورة الـ Scale حتى تتلائم. يمكنك عندئذ حفظ الصورة التي تم تحويلها. إن هذه هي أسهل طريقة وجدتها لتقليص الـ glyph متعدد الصور ليصبح bitmap لصورة واحدة.

بالقطع يمكنك أيضاً إنشاء الـ glyphs الخاصة بك باستخدام أى bitmap editor في أى من الحالتين، يمكنك الآن إدخال الـ glyphs الخاصة بك في الـ SpinButton. على سبيل المثال، جرب هذه الخطوات:

١- أضف الـ SpinButton من الـ Samples page tab على الـ VCL palette. تأكد من إختيار الـ SpinButton وليس الـ SpinEdit.

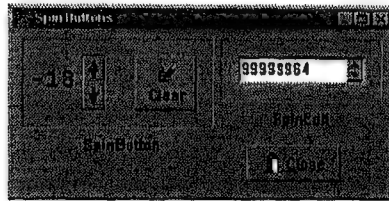
٢- اضغط الزر البيضاوى لخاصية الـ DownGlyph لفتح الـ Image Editor. قم بتحميل ملف صورة سهم سفلى معدل مثل الـ Arrow1d.bmp.

٣- أعد الخطوة الثانية مع خاصية الـ UpGlyph لتحميل الـ Arrow1u.Bmp معدل.

٤- اعتماداً على حجم glyph bitmap، قد يكون عليك أن تعدل أبعاد الـ SpinButton لإظهار الـ glyph كاملة.

## استخدام ال SpinEdit components:

على القرص المدمج: إن ال SpinEdit يجمع بين SpinButton و Edit input field لإنشاء زر إسطوانى ذا إمكانيات editing كاملة، بما فى ذلك أوامر القطع والنسخ واللصق. يوضح شكل (٦-١٢) ال SpinEdit و SpinButton من تطبيق ال SpinButt على القرص المدمج فى دليل ال Source\SpinButt. Source. توضح القائمة (٦-٢) ال source code للتطبيق.



شكل (٦-١٢): التطبيق SpinButt يوضح SpinEdit object وال SpinButton

**ملحوظة:** لقد استبدلت ال SpinEdit object وال SpinButton بـ UpDown الجديد نسبياً. قد تريد استخدام ال UpDown بدلاً من ال SpinButton وال SpinEdit، إلا إذا كنت تستخدم نسخ أولية من Delphi، أو إذا كنت تحتاج أن تقدم تجانس خلفى. ولكن، لا يجد شئ خطأ فى هذه ال components ولا يجب أن تتردد فى استخدامها إذا وفرت لك الخصائص التى تريدها. بعد الفصل القادم، سوف اشرح كيفية استخدام ال UpDown.



### القائمة (٦-٢): Spinbutt\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, Spin, StdCtrls, ExtCtrls, Buttons;

type
  TMainForm = class(TForm)
```

## Attaching Buttons and Check Boxes: الباب السادس

---

```
SpinButton1: TSpinButton;
  Label1: TLabel;
  SpinEdit1: TSpinEdit;
  SpinEdit: TLabel;
  SpinLabel: TLabel;
  BitBtn1: TBitBtn;
  Bevel1: TBevel;
  Bevel2: TBevel;
  BitBtn2: TBitBtn;
  procedure FormCreate(Sender: TObject);
  procedure SpinButton1DownClick(Sender: TObject);
  procedure SpinButton1UpClick(Sender: TObject);
  procedure BitBtn1Click(Sender: TObject);
  procedure SetSpinButtonCaption;
private
  { Private declarations }
  Count: Integer;
public
  { Public declarations }
end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

const
  minCount = -99;
  maxCount = 99;

procedure TMainForm.SetSpinButtonCaption;
begin
  SpinLabel.Caption := IntToStr(Count);
```

end;

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
    Count := 0;
end;
```

```
procedure TMainForm.SpinButton1DownClick(Sender: TObject);
begin
    if Count > minCount then Dec(Count);
    SetSpinButtonCaption;
end;
```

```
procedure TMainForm.SpinButton1UpClick(Sender: TObject);
begin
    if Count < maxCount then Inc(Count);
    SetSpinButtonCaption;
end;
```

```
procedure TMainForm.BitBtn1Click(Sender: TObject);
begin
    Count := 0;
    SetSpinButtonCaption;
end;
```

end.

إن الـ SpinEdit له شكل Button داخلي من نوع الـ SpinButton . (إن الاسم Button يعتبر إختيار سيء- إنه ليس Button component ، ولكن object من الـ SpinButton يدعى Button) . استخدم الـ Button الخاص بالـ SpinEdit كما تفعل في حالة الـ SpinButton المنفرد . على سبيل المثال ، لتعيين صور glyph علوية وسفلية من نوع الـ TBitmap ، يمكنك إدخال عبارات مثل هذه :

```
SpinEdit1.Button.UpGlyph := NewUpGlyph;
SpinEdit2.Button.DownGlyph := NewDownGlyph;
```

## ال Static Text أو (TStaticText):

إن Delphi يقدم component جديد نسبياً لعرض ال labels، وهو ال StaticText، على Additional palette. ومثل ال Standard Label، يعتبر ال StaticText مفيداً في إنشاء ال interactive labels يمكن للمستخدم أن يضغطها بالفأرة.

**ملحوظة:** من الناحية الفنية، إن TStaticText class تشبه ال TLabel، فيما عدا أن ال TStaticText تنحدر من ال TWinControl. بمعنى آخر، إن ال Label ليس لديها window handles؛ ولكن ال StaticText لديها. في حالة ضرورة أن يتمي ال accelerator key control الى ال window handle، فعليك استخدام ال StaticText بدلاً من ال Label.

إن ال StaticText له ثلاث خصائص غير موجودة في ال Label. والخصائص الجديدة هي:

● **BorderStyle:** حدد هذه بـ sbsNone، أو بـ sbsSingle، أو بـ sbsSunken. بإستثناء ال sbsNone، قد تريد تحديد ال AutoSize بـ False؛ وإلا؛ يتم تحديد حجم ال control تلقائياً اعتماداً على كم نص ال Caption الذي يعرضه.

● **TabOrder:** حدد هذه الخاصية حسب ال tab order لبند ال StaticText. ولأن ال component له ال window handle، فإنه قد يتلقى ال input focus (بالرغم من أن المستخدمين لا يستطيعون الكتابة داخله).

● **TabStop:** حدد هذه بـ True لإضافة ال StaticText الى قائمة الآخرين الذين يتلقون ال input focus، عندما يضغط المستخدمون ال Tab.

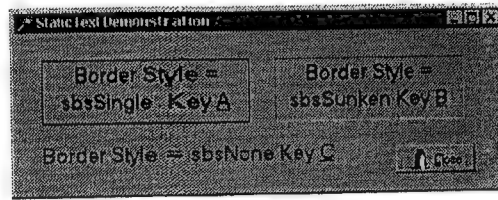
إثنين من انماط ال Label غير موجودين في ال StaticText وهما:

● **Transparent:** عندما تحدد هذه الخاصية بـ True، فإن خلفية ال Label لا تغير أي control أو نافذة أخرى. وبالإرتباط بخاصية ال Color، يمكنك تحديد

ال Transparent بـ False لتلون الخلفية بوضوح (والذى يعرض النص اسرع عن كون ال Transparent محدد بـ True). ان ال StaticText لا تكون ابداً Transparent، ولذلك فهى تفتقد هذه الخاصية. ولكن، لإنشاء StaticText يبدو شفافاً، ببساطة حدد خاصية ال Color له بنفس الطريقة وكان object ما موجود تحت النص.

• **WordWrap** : مع ال Label، يمكنك اختيارياً ان تحدد هذه الخاصية بـ True لقطع الخطوط الطويلة من النص بين الكلمات تلقائياً. ودائماً ما تقوم ال StaticText بضبط النص الطويل داخل حجمه المحدد. حدد ال AutoSize بـ False عند عرض نص طويل فى ال StaticText؛ والا، لن يضبط ال control النص الخاص به ولكن بدلاً من ذلك سوف يعرض خطأ واحداً، بغض النظر عن طوله.

**على القرص المدمج:** ان ال StaticText يفيد فى إنشاء text objects يمكن للمستخدمين اختيارها بالضغط بالفأرة. يوضح تطبيق ال StaticDemo فى القائمة (٦-٣) كيفية فعل هذا، ويظهر أيضاً الانماط الثلاثة المختلفة لـ StaticText. يوجد البرنامج على القرص المدمج فى دليل ال Source\StaticDemo. يوضح الشكل (٦-١٣) الناتج عن تنفيذ البرنامج.



شكل (٦-١٣): يظهر تطبيق ال StaticDemo انماط ال StaticText، كما يظهر كيفية إنشاء عناصر نص يمكن للمستخدمين ضغطها بالفأرة

القائمة (٦-٣): taticDemo\Main.pas

```
unit Main;
```

```
interface
```

## Attaching Buttons and Check Boxes: الباب السادس

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, StdCtrls, Buttons;

type

```
TMainForm = class(TForm)
    StaticText2: TStaticText;
    StaticText1: TStaticText;
    StaticText3: TStaticText;
    BitBtn1: TBitBtn;
    procedure StaticText1MouseDown(Sender: TObject;
        Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure StaticText1MouseUp(Sender: TObject;
        Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure StaticText1Click(Sender: TObject);
```

private

```
{ Private declarations }
public
{ Public declarations }
end;
```

var

MainForm: TMainForm;

implementation

{ \$R \*.DFM }

var

SavedStyle: TStaticBorderStyle;

{ Change static text border style to indicate its selection }



```

procedure TMainForm.StaticText1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  with Sender as TStaticText do
    begin
      SavedStyle := BorderStyle;
      if BorderStyle = sbsSunken
        then BorderStyle := sbsSingle
        else BorderStyle := sbsSunken;
      end;
    end;
end;

```

```

{ Reset static text border style when mouse button released }
procedure TMainForm.StaticText1MouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  with Sender as TStaticText do
    BorderStyle := SavedStyle;
  end;
end;

```

```

{ Make a sound when text object is clicked }
procedure TMainForm.StaticText1Click(Sender: TObject);
var
  S: String;
begin
  MessageBeep(0);
  S := TStaticText(Sender).Name;
  ShowMessage('You selected ' + S);
end;

end.

```

## الباب السادس: Attaching Buttons and Check Boxes

يوضح برنامج العرض إحدى الطرق لإعطاء إرجاع بصري عندما يضغط المستخدمون. StaticText. لقد تم تعيين MouseDown و MouseUp ل StaticText الثلاثة في الـ form، كلاً منها له قيمة خاصة بـ BorderStyle مختلفة.

عندما يضغط المستخدم الفأرة على أحد الـ StaticText، يحفظ MouseDown handler الـ BorderStyle الحالي في global variable ثم يغير النمط إلى شيء آخر. عندما يتلقى المستخدم تأكيداً بصرياً بأن الـ control يتعرف على الضغط على الفأرة. عندما يطلق المستخدم الفأرة، يعيد MouseUp تحديد خاصية الـ BorderStyle بقيمتها المحفوظة، والتي تستعيد المظهر الأصلي للـ control.

في كلتا الحالتين، من الضروري أن تستخدم عبارة مثل التالية:

```
with Sender as TStaticText do
    BorderStyle := SavedStyle;
```

أن Sender parameter الذي تم تمريره للـ event handler يساوي الـ object الذي ضغطه المستخدم. ولأن الـ form أيضاً لها خاصية BorderStyle، والتي هي من نوع آخر، من الضروري أن تستخدم عبارة with أو أى مشير آخر لتخبر الـ compiler أى من خاصية الـ BorderStyle تستخدم. والطريقة لفعل هذا هو أن تستخدم عبارة مثل:

```
StaticText1.BorderStyle := sbsSunken;
```

بدون اسم الـ object والنقطة، يعتبر الـ BorderStyle غامضاً عند استخدام خاصية الـ BorderStyle في عبارة برنامج، إذا تلقيت رسالة الخطأ "Incompatible types: 'TFormBorderStyle' and 'TStaticBorderStyle'", استخدم إحدى التقنيتين السابقتين لحل التداخل.

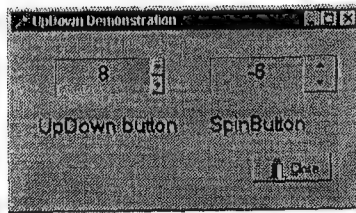
والـ event handler الثالث والأخير في التطبيق يعطى نغمة للإشارة إلى أن الـ StaticText قمتم اختياره، ويعرض أيضاً رسالة تحتوي على اسم الـ object. وهذا يظهر كيف يمكنك استخدام خاصية الـ Sender لتحديد أى بند من عناصر الـ StaticText قام المستخدم باختياره. أو، يمكنك برمجة OnClick منفصلة لكل StaticText.

## : Up-Down Button Controls

وهناك component آخر جديد نسبياً يجعل من السهل إنشاء أزرار ذات اسهم علوية وسفلية، والتي تستخدم لاختيار قيم أو تحريك مجموعة من العناصر في edit window أو label وكانت الطريقة الوحيدة لفعل هذا في النسخ الأولى من Delphi هي باستخدام الـ Sample SpinButton، الموضح سابقاً في هذا الباب. ولكن، لأن هذا مجرد Sample، وليس من المعايير الرسمية لـ Delphi، كان المبرمجين غير مستعدين لاستخدامه خوفاً من أن الـ Borland قد لا يدعم الـ SpinButton في المستقبل.

وإننى أؤكد أن الـ UpDown على الـ Win32 يفوق الـ SpinButton، وإننى أرشح استخدام الـ UpDown من الآن فصاعداً. فإنه يسهل ربطهم بالـ controls الأخرى مثل الـ StaticText والـ Edit، وكـ Windows controls أصلية، فإنهم يصفون مظهراً متماشياً مع واجهات تطبيق البرمجيات الأخرى.

**على القرص المدمج:** لمقارنة الـ UpDown و الـ SpinButton كتبت تطبيق الـ UpDown على القرص المدمج. يوضح شكل (٦-١٤) عرض التطبيق. توضح القائمة (٦-٤) الـ source code. افتح ملف مشروع الـ UpDownDemo.dpr واضغط F9 لتشغيل البرنامج في Delphi. اضغط الـ الأزرار إلى اليسار من عناصر الـ StaticText يربط الـ UpDown تلقائياً بالـ text object الخاص به؛ يتطلب الـ SpinButton برمجة ليفعل نفس الشيء.



شكل (٦-١٤): يقارن تطبيق الـ UpDownDemo الـ UpDown الأصلي في الـ Win32 بـ SpinButton الخاص بـ Delphi على Samples palette.

القائمة (٦-٤): UpDownDemo\Main.pas

```
unit Main;
```

## Attaching Buttons and Check Boxes: الباب السادس

---

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, StdCtrls, Spin, ComCtrls, Buttons;

type

```
TMainForm = class(TForm)
    UpDown1: TUpDown;
    SpinButton1: TSpinButton;
    Label1: TLabel;
    Label2: TLabel;
    BitBtn1: TBitBtn;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    procedure SpinButton1UpClick(Sender: TObject);
    procedure SpinButton1DownClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;
```

var

```
MainForm: TMainForm;
```

implementation

```
{ $R *.DFM }
```

{ These constants and the two event handlers are needed by SpinButton components to associate them with another control, a StaticText object in this demonstration. The new UpDown component needs none of this programming because

it can be automatically associated with another control. }

const

Min = -10;

Max = +10;

{ Respond to user clicking the SpinButton's Up button }

procedure TMainForm.SpinButton1UpClick(Sender: TObject);

var

V: Integer;

begin

V := StrToInt(StaticText2.Caption);

if V = Max

then V := Min

else inc(V);

StaticText2.Caption := IntToStr(V);

end;

{ Respond to user clicking the SpinButton's Down button }

procedure TMainForm.SpinButton1DownClick(Sender: TObject);

var

V: Integer;

begin

V := StrToInt(StaticText2.Caption);

if V = Min

then V := Max

else dec(V);

StaticText2.Caption := IntToStr(V);

end;

end.

## الباب السادس: Attaching Buttons and Check Boxes

لا شيء من البرمجة الموجودة في القائمة (٦-٤) يحتاج إليه مع UpDown controls- فهم يسهل ربطهم مع ال text labels وال edit field. على سبيل المثال، في هذه الحالة، لربط ال UpDown مع عنصر النص الخاص به على اليسار، فقد حددت خاصية ال Associate لـ UpDown1 بـ StaticText1. ويتولى ال control أمر جميع تفاصيل العرض الخاصة بتحويل القيم العددية الى strings وعرضها في نافذة النص. هذه الخطوات تتم يدوياً لـ SpinButton كما هو موضح في القائمة:

### خصائص ال UpDown:

ويقدم ال UpDown ايضاً عدداً من الخصائص الممتعة التي يمكنك تجربتها واهم هذه الخصائص هي:

• **AlignButton**: حدد هذه الخاصية بـ udLeft أو udRight للصق الازرار باحد جوانب أو الجانب الآخر لل control المرتبط بها. وهذه الخاصية ليس لها تأثير الا اذا كانت ال Associate محددة باسم ال control الآخر. عند استخدام هذه الخاصية، لن تنجح محاولات تغيير موضع ازرار ال UpDown يدوياً بالنسبة لل control المرتبط بها.

• **ArrowKeys**: عند ما تحدد بـ True، تقوم هذه الخاصية بتشغيل مفاتيح الاسهم المشيرة لاعلى ولأسفل في لوحة المفاتيح لضغط ازرار ال control يجب لل UpDown ان يكون input focus (حدد ال TabStop بـ True).

• **Associate**: حدد هذه الخاصية باسم ال object الآخر، والذي سيكون في الغالب Label، أو StaticText، أو Edit. استخدم ال AlignButton لوضع ال UpDown على احد جانبي ال control المرتبط به. ان ربط control بـ UpDown هو كل ما تحتاج فعله لتوفير تتابع آلى للقيم في ال control.

• **Cursor**: اختر شكل مؤشر اذا كنت تريد ان يتغير عندما يوجه المستخدمين الفأرة الى ازرار ال UpDown.

• **Increment**: في كل مرة بضغط المستخدم زراً تتغير قيمة ال UpDown على حسب القيمة المحددة في ال Increment.

● **Max, Min** : ضع حداً لمدى ال control بتحديد هاتين القيمتين . فى العادى ، يجب ان تكون ال Min أقل من ال Max ؛ ولكن ، اذا كانت ال Min أكبر من ال Max ، يبدو ال UpDown وهو يعمل الى الوراء (اضغط الزر العلوى مثلاً يخفض قيمة ال control ) اذا لم تكون تريد إرباك مستخدمى برنامجك ، تأكد من ان قيمة ال Min أقل من ال Max .

● **Orientation** : بالرغم من انه يحمل اسم ال UpDown ، يمكن ان يعرض ال component ازراره افقياً (ال udHorizontal) أو رأسياً (ال udVertical) . استخدم أى تحديد تريده .

● **Position** : ان خاصية ال Position تساوى القيمة الحالية لـ ال UpDown . (اذاً ، لماذا لم يسموها Value ؟) اقرأ هذه الخاصية فى وقت التشغيل لتحديد القيمة التى تم اختيارها حالياً .

● **Thousands** : حدها بـ True اذا كنت تريد علامات ترقيم بين مجموعات مكونة من ثلاثة خانات رقمية فى ال controls التى لها قيم ال Position تساوى اربعة ارقام أو أكثر .

● **Wrap** : عندما تحدد هذه الخاصية بـ False ، تتوقف الزيادة والنقصان عندما يصل ال control الى قيمته القصوى أو الصغرى . عندما تحدد هذه الخاصية بـ True ، تقل القيمة . على سبيل المثال ، اذا كانت ال Max تساوى 100 ، فإن محاولة زيادة ال control ليصبح 101 تجعله يرجع الى قيمة ال Min .

### الاستجابة الى ال events :

يمكن لازرار ال UpDown ان تستجيب لبعض ال events ولكن فى الغالب تكون الحاجة الى اثنين فقط : ال OnChanging وال OnClick . ويستدعى ال Delphi ال OnChanging عندما تكون قيمة ال Position ال UpDown على وشك ان تتغير . ويتم تعريف ال event handler procedure فيما يشبه هذا :

```
procedure TMainForm.UpDown1Changing(Sender: TObject;
var AllowChange: Boolean);
```

## الباب السادس: Attaching Buttons and Check Boxes

كما هو معتاد، يشير الـ Sender الى الـ component الذي اطلق الـ event. في داخل الـ event handler، حدد الـ AllowChange بـ True لتشغيل الـ UpDown control لتغيير قيمة الـ Position. حدد الـ AllowChange بـ False لإلغاء عمل الـ cancel، ولعدم لتغيير قيمة الـ Position. في الغالب، إنك سوف تستخدم برنامج الـ event handler هذا لتضع حدوداً واضحة على قيمة الـ Position، وكذلك عندما يكون الـ UpDown غير مرتبط بـ cancel آخر. على سبيل المثال، بدلاً من إبطال الـ UpDown، قد تستخدم قيمة الـ check Box لتحديد اذا ما كنت سوف تسمح لمستخدمين بضغط الـ cancel، . يمكن ان تكتب برمجة الـ OnChanging كما يلي:

```
begin
  if CheckBox1.Checked
    then AllowChange := True
    else AllowChange := False;
end;
```

الـ event الثاني لـ UpDown الذي ترغب في استخدامه هو الـ OnClick. وهذا الـ event يأتي قبل الـ OnChanging. استخدم الـ OnClick للاستجابة لاختيار المستخدم لأزرار الأسهم التابعة للـ control. ويتم تعريف الـ event handler procedure كما يلي:

```
procedure TMainForm.UpDown1Click (Sender: TObject;
  Button: TUDBtnType);
```

يشير الـ Sender الى أى من أزرار الـ UpDown تم إختياره في حالة تشارك controls متعددة نفس الـ event handler. يشير الـ Button parameter الى أى من زرى السهمين قام المستخدم بضغطه عندما يكون الـ Button مساوياً لـ btNext، يكون المستخدم قد ضغط السهم الأيمن أو العلوى؛ عندما يكون الـ Button مساوياً لـ btPrev، يكون المستخدم قد ضغط السهم الأيسر أو السفلى.

قد تستخدم الـ event handler OnClick لأداء بعض العمليات قبل أن تتغير قيمة الـ UpDown. تذكر أن الـ OnClick يطلق قبل الـ OnChanging، وإنه فقط عندما يرجع الـ OnChanging الـ AllowChange مساوية لـ True تتغير قيمة الـ Position الى أعلى أو أسفل بالكمية المحددة في النقصان.



## افكار للمستخدم الخبير

• يمكن أن يكون للـ `GroupBox` قائمة `PopupMenu` منفصلة عن الـ `PopupMenu` الخاصة بالـ `from`. قم بتعيين `PopupMenu` لخاصية الـ `PopupMenu` التابعة للـ `GroupBox`، بنفس الطريقة التي قمت بها بتخصيص الـ `PopupMenu` للـ `form`. إن استخدام `GroupBox` متعددة في الـ `form` يعد طريقة جيدة لتصميم `floating pop-up menus` متعددة تعتمد على أين يضغط المستخدم زر الفأرة الأيمن.

• لإنشاء أزرار تظهر وتختفي إستجابة لأوامر، أو ظروف البرنامج المتنوعة، حدد خاصية الـ `Visible` للزر بـ `True` أو `False`. يمكنك أن تفعل هذا لأي من الـ `components` التي تم شرحها في هذا الباب، وكذا غالبية الـ `Delphi` `components` الأخرى.

• اضغط مرتين أى قيمة فى خاصية الـ `Color` (اضغط القيمة الموجودة على اليمين؛ وليس الاسم) لفتح نافذة الـ `Color Editor`. اختر أى لون، أو استخدم زر الـ `Define Custom Colors` لفتح جزء آخر من الـ `editor` الذى يسمح ببرمجة ألوان مخصصة باستخدام قيم الأزرق، والأخضر، والأحمر، أو بدلاً من قيم البريق، الإمتصاص، واللون.

• أدخل نصاً فى خاصية الـ `Hint` للـ `SpeedButton`، وحدد الـ `ShowHint` بـ `True`. يرى المستخدمون عندئذ الـ `hint box` صغيراً عندما يضعوا مؤشر الفأرة على الزر.

• استخدم خاصية الـ `Spacing` الخاصة بالـ `BitBtn` لتعديل كمية المسافة بين صورة الـ `glyph` وتعليق الزر. إن القيمة الافتراضية حسب النظام هى 4. استخدم 1- لوضع النص فى المنتصف بين الصورة وحافة الزر. حدد الـ `Spacing` بـ 0 إذا لم تكن تريد مزيداً من `pixels` بين الصورة والنص. إن الـ `SpeedButton`، الذى ليس له `Caption`، له أيضاً خاصية `Spacing` يمكنك استخدامها لتعديل موضع الـ `glyph`.

## الباب السادس : Attaching Buttons and Check Boxes

• بالرغم من أن ال Panels تعتبر مفيدة للغاية ك objects بصرية خالصة، إلا إنه يمكنك إنشاء event handlers لها. على سبيل المثال، قم بإنشاءOnClick event handlers لأداء عمل عندما يضغط المستخدمون Panel object. و event آخر غاية في الإفادة هو ال OnResize، والذي يستخدم بشكل مثالي لل Panels المحاذية في ال client area بالنافذة عندما يقوم المستخدمون بإعادة تحديد حجم النافذة، يمكن لل OnResize event handler الخاص بال Panel أن يعدل مواضع ال controls داخل النافذة.

• إن ال SpinButton الذي تم وصفه في هذا الباب يستخدم object من ال TTimerSpeedButton class الغير موثوقة. وبناء على ال TTimerSpeedButton class، تقوم ال TTimerSpeedButton class بإنشاء ال Timer الذي يحاكي تكراراً لضغطه الزر عندما تكون ضاغطاً زر الفأرة. يفعل ال component هذا بإستدعاء ال Click method الخاصة به، والذي يشبه ال OnClick. أنظر ملف ال Spin.pas في دليل ال Source\Samples الخاص ب Delphi للحصول على ال source code ل ال TTimerSpeedButton class.

• يمكن لأي control هاو أن يجمع radio buttons، و check boxes و component objects الأخرى. على سبيل المثال، اذا اردت مجموعتي radio buttons دون حدود، استخدم ال TPanel كحاويات وأغلق حدودها.

• استخدم ال OnClick وال OnChanging مع ال UpDown عندما لا يكون مرتبط ب control آخر. في المعتاد، عندما يكون ال UpDown مرتبطاً ب control text أو edit، تكون ال events غير لازمة.

## المشروعات التي يمكنك تجربتها

(٦-١): اكتب برنامجاً يعرض Bevel و Panel في تشكيلات عديدة ممكنة. سوف تجد هذا نافعاً كمرشد لك لإنشاء Panels و Bevels جديدة في التطبيقات. سوف يسمح برنامجك للمستخدمين بإدخال خصائص ال objects ورؤية تأثيراتها في الحال.

- (٢-٦): قم بتعديل Calc32 لاستخدام القيم الصحيحة ٦٤ بت .  
ملحوظة : استخدم نوع البيانات Int64 .
- (٣-٦): قم بإنشاء glyphs للـ Calc32 SpeedButtons وبشكل عام ،  
رتب واجهة تطبيق المستخدم الخاصة بالآلة الحاسبة .
- (٤-٦): تقنية التحريك المقترحة فى هذا الباب . أولاً قم بجمع ملفات الـ  
Dooropen.bmp والـ Doorshut.bmp باستخدام الـ Image Editor الخاص بـ Delphi . يجب ان يحتوى الملف الناتج اربع  
صور glyph منفصلة ، كلاً منها لها حجم ١٦×١٦ pixels . قم  
بتعيين الـ glyph التى تم تعديلها لـ BitBtn وقم بتشغيل البرنامج -  
يغلق الباب عندما تضغط الزر . قد تريد ان تبدأ مكتبة من صور الـ  
glyph التى تم تحريكها .
- (٥-٦): اكتب برنامج glyph catalog (Glyphlst) يعرض كل انماط الـ  
glyph الموجودة فى دليل الـ Images\Buttons الخاص بـ  
Delphi .
- (٦-٦): اكتب برنامجاً يعرض اربعة SpeedButton توضح الحالات  
الممكنة لـ glyph bitmap الخاصة بها (استخدم خصائص الـ  
AllowAllUp ، Down ، و Enabled ) . قم بتعيين glyph  
الـ bitmap للـ SpeedButtons فى وقت التشغيل ، ربما يكون قد تم  
اختياره من دليل ملف .
- (٧-٦): متقدم . اكتب floating-point calculator باستخدام الـ  
Calc32 كمرشد لك لاستخدام انواع البيانات الـ Double  
(8 Byte) أو Extended (10 Byte) لتخزين القيم داخلياً .

### ملخص:

- يقدم الـ Windows ثلاثة انواع من الازرار المعيارية push buttons ،  
radio buttons ، check boxes . يقوم Delphi بإحتواء هذه الـ controls  
المعيارية فى الـ Button ، CheckBox ، و RadioButton .

## الباب السادس : Attaching Buttons and Check Boxes

• استخدم Components array الخاص بال form للوصول الى جميع ال objects- على سبيل المثال ، لإبطال كل button control إلا واحد فقط فى نافذة .

• تعمل ال CheckBoxes فى المعتاد on/off toggles ، ولكن يمكن أيضاً أن يعملوا three-way switches وذلك بتحديد ال AllowGrayed ب True . يمكن للمستخدمين حيثذ الرجوع للمربع لتشغيل ، وإبطال ، وجعل علامة الصح الموجودة به رمادية .

• يمكنك تلوين CheckBox وال RadioButton control ، ولكن بسبب حداً ما فى ال Windows ، لا يمكنك تلوين ال Button . للحصول على زر أكثر لوناً ، استخدم ال BitBtn .

• يمكن لـ BitBtn وال SpeedButton ان تعرض bitmap glyphs والتي تمثل هدف ال control ان ال glyph هو Windows bitmap تم تقديمه ك TBitmap ، وهو فى المعتاد ١٦×١٦ pixels ذو ستة عشر لوناً . ولكن ، لا يوجد حدوداً على حجم ال glyph . ويمكن لل glyph ان يكون لها من واحد الى اربع صور منفصلة تمثل الحالات المختلفة للزر .

• تظهر ال SpeedButton عادة فى toolbars (انظر الباب التالى) ، ولكن كما يوضح تطبيق ال Calc32 الخاص بهذا الباب ، يمكنك ايضاً استخدام ال SpeedButton ك objects مستقلة .

• استخدم ال GroupBox لتجميع ال controls المتعددة بطريقة منطقية . غالباً ما يحتوى ال GroupBox على RadioButton ، ولكن يمكن ان يحتوى ايضاً controls اخرى يمكن للمستخدمين ضغط ال Tab لتحريك ال focus من احد ال GroupBox الى آخر .

• استخدم ال GroupBoxes كاسلوب أبسط لإنشاء مجموعات من ال RadioButton . أضف ال button label فى خاصية ال Items للـ RadioGroup .

• ال Panel وال Bevel عبارة عن visual component ، ولكنها تفيد فى تنظيم شاشة مزدحمة ممتلئة بالازرار وال controls الأخرى .

• ان Samples page tab على ال VCL palette يقوم button controls  
إضافيين ، وهما ال SpinButton وال SpinEdit . قم بتعيين ال glyphs الخاصة  
بك لخصائص ال DownGlyph وال UpGlyph الخاصة بهذه ال objects لعرض  
اسهم علوية وسفلية أو objects أخرى مخصصة .

• استخدم ال StaticText لإنشاء labels ذات window handles . يعتبر  
ال StaticText نافعا أيضاً فى إنشاء مساحات نص يمكن للمستخدمين ضغطها  
كالأزرار .

• استخدم ال UpDown بدلاً من ال SpinButton وال SpinEdit . لربط  
الاسهم العلوية السفلية بال label objects ، و static text ، edit objects .

يوضح الباب التالى كيفية استخدام ال SpinButton وال Panel لإنشاء  
toolbars و status - عنصرى واجهة التطبيق الذان يجب ان يتوفران فى كل تطبيق  
Windows . وسوف نلقى نظرة . ايضاً على اختراع جديد نسبياً وهو  
ال Coolbars .

\*\*\*\*\*

# الباب السابع

## إنشاء الـ Toolbars ، الـ Coolbars ، والـ Status Panels

محتويات هذا الباب:

• Components

• Toolbars

• Status Panels

• Coolbars

• StatusBars

ان إحدى علامات الجودة فى برنامج الـ Windows المكتوب بشكل جيد هى الـ toolbar للـ speed button لاختيار الاوامر بالفأرة بسرعة. وتمثل الـ Toolbar buttons للفأرة مائثله المفاتيح الهامة للوحة المفاتيح- اختصارات للمستخدمين الذين لا يريدون إضاعة الوقت فى فتح قوائم لاختيار أوامر. يوضح هذا الباب كيفية إنشاء الـ toolbars فى تطبيقات Delphi.

ويوضح هذا الباب ايضاً كيفية إنشاء الـ Coolbars ، والتي يمكن ان تحتوى على أكثر من bands التي يمكن ان تحمل Window control . على سبيل المثال ، يمكن ان يعرض الـ Coolbar مجموعة من الازرار drop-down list ، check ، boxes ، Internet hot-link buttons ، و controls اخرى . (اذا كان لديك Microsoft's Internet Explorer ، فإنك تعرف الـ Coolbars- هناك واحد أعلى قمة نافذة الـ Explorer يعرض browser buttons ومنتقى عنوان الـ URL). يمكن للمستخدمين ضغط وسحب الفأرة لإعادة تنظيم لوحة الـ Coolbars .

وهناك عنصر واجهة تطبيق وهو ال status panel ، والتي تظهر غالباً في اسفل نافذة ما ، وغالباً ما تكون مقسمة الى قطاعات عديدة . على سبيل المثال ، يمكن ان تعرض ال status panel الخاصة بال text editor ارقام العمود والخط لموضع المؤشر الحالي . أو ، يمكن أن تعرض ال panels التاريخ والموعد لتذكير متابعي الساعة كما تأخروا عن الموعد النهائي . ويوضح هذا الباب طرق مختلفة لإنشاء ال status panel باستخدام ال Delphi components ، وكذا ال StatusBar مع ال Win32 .

**ملحوظة:** لأن ل Delphi طرق مختلفة لإنشاء ، ال toolbars وال status panels (ولكن هناك طريقة واحدة لإنشاء ال Coolbars) ، فإن هذا الباب يستخدم المصطلحات العامة ال toolbar ، Coolbar ، status panels لهذه العناصر . والكلمات ال Toolbar و Coolbar و StatusBar تشير إلى ال Win32 component الخاصة ب Delphi .

## Components:

فيما يلي بعض ال Delphi components لإنشاء ال toolbars وال Coolbars و ال status panels .

● **Animate** : غالباً ما تحتوي ال Coolbars على ال Animate لتقديم بعض الصور الجرافيكية التي تم تحريكها لعرض إذ كان بدونها سيصبح عرضاً ثابتاً مملاً . ويوضح مظهر ال Coolbars الخاص بهذا الباب كيفية إنشاء ايقونة جذابة تم تحريكها ، والتي لها هدف عملي وهو جعل المستخدمين يعرفون ان البرنامج يعمل بايجابية . Win32 : Palette .

● **CoolBand** : وال Objects الخاصة بهذا النوع من ال components ، والتي لا توجد على VCL palette ، يتم تخزينها في ال Coolbar ويقوم هو بالتحكم فيها . وكل CoolBand يمكن ان يحمل Coolbars آخر مثل ال Toolbar أو ال Palette . ComboBox . لا يوجد .

● **Coolbar** : يشبه toolbar في النمط ، ولكنه أكثر تعقيداً في التشغيل ، وبعد ال Coolbar object متعدد القطاعات مقسم الى مساحات تسمى ال bands (انظر ال

## الباب السابع : إنشاء الـ Toolbars ، الـ Coolbars ، والـ Status Panels

windows CoolBar الـ wrapper الـ Coolbar صفة الـ CoolBand control والذي يمكنه عرض وإدارة أى عدد من الـ windowed controls (وبخاصة ، تلك الـ components التى تنحدر من الـ TWinControl). يمكن للمستخدمين تشكيل الـ Coolbars اثناء التشغيل بالضغط والسحب بالفأرة . Win32 : Palette .

● **Panel** : استخدم هذا الـ component لإنشاء مسطح لإقامة toolbar و status panel . يمكنك تغيير مظهر الـ 3D للمسطح لنجعله يبدو مثل الفجوة المفرغة أو الهضبة المرتفعة . عند استخدام الـ panel كـ toolbar ، فإنه يحمل SpeedButtons ولكن يمكن ان تعمل الـ panel كحاوية لاغراض عامة يمكنها حمل panels اخرى ونصوص لإقامة مساحات عرض قطاعية . Palette : Standard .

● **SpeedButton** : يوضح هذا الباب كيفية استخدام الـ SpeedButtons لإقامة toolbars باستخدام الـ Panel . راجع ايضاً الباب السابق لمزيد من المعلومات حول الـ events وخصائص الـ SpeedButton . الـ Palette : Additional .

● **StatusBar** : إحدى طرق إنشاء status panel هى استخدام الـ StatusBar ، والـ StatusBars ، التى تظهر بالضبط فى اسفل النافذة ، تحتوى واحداً أو أكثر من الـ StatusBarPanel . Win32 : Palette .

● **StatusPanel** : وهذا الـ component . يشكل قطاعاً ، يسمى Panel ، من الـ StatusBar ويتم عرضه والتعامل معه من قبل الـ StatusBar . (لا تخط بين هذا الـ component والـ component الأكثر عموماً Panel) : الـ Palette : لا يوجد .

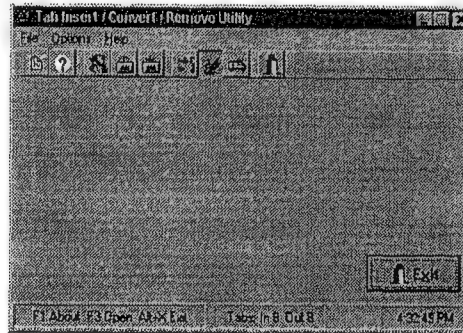
● **ToolBar** : يقدم هذا الـ component طريقة بديلة لإنشاء الـ ToolBar فى نافذة تطبيق والذي يستطيع عرض الـ ToolButtons بأساليب متعددة . استخدم الـ ToolBar لتمنح تطبيقاتك مظهر يشبه متصفح Web ولتواكب إرشادات واجهة التطبيق للـ Windows 98 . Win32 : Palette .



• **ToolButton**: يمكن لل **ToolBar** ان يعرض ويحتفظ بواحد أو أكثر من ال **ToolButton**، والتي تشبه ال **SpeedButton**. مثل ال **SpeedButton**، يمكن لل **ToolButtons** ان تعرض ايقونات، ولكن يمكنها ايضاً عرض **text labels**. وكذلك، يمكن ان يكون لل **ToolButtons** صور مرتبطة عديدة تغير مظهرها عندما يشير المستخدم الى أو يضغط زراً. **Palette**: لا يوجد.

## ال **Toolbar**:

**على القرص المدمج**: ان تطبيق ال **Tabs application** فى هذا الباب يوضح تقنيات ال **toolbar** و **status panel**، يوضح شكل (٧-١) مثال لل **toolbar for Tabs**. بالرغم من ان ال **source code** للبرامج طويلة جداً لكى تذكر هنا بأكملها، توجد جميع الملفات على القرص المدمج فى دليل ال **Source\Tabs**. افتح ملف مشروع **Tabs.dpr** لفحص ملفات التطبيق. هذا نموذج لبرنامج يستخدم **units** متعددة، جميعها مرتبطة بـ **visual form** ال واحدة. على سبيل المثال، ان ال **Inidata unit** فى دليل ال **Source\Inidata** تقدم خدمات ملف ابتدائية لتطبيق ال **Tabs**.



شكل (٧-١): تطبيق ال **Tabs** يحذف ويضيف **tab control codes** فى ملفات النص، ويمكنه ايضاً تحويل **tab interval settings**

يعتبر تطبيق ال **Tabs** أكثر من مجرد برنامج عرض - فهو **Utility** مفيد قد تود ان تحتفظ به على قرصك لفترة اطول بعد الانتهاء من هذا الكتاب. استخدم ال **Tabs** لإدخال وحذف **tab control codes** من ملفات النص، ولتحويل الملفات

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

ذات tab مختلفة. على سبيل المثال، يمكن للـ Tabs ان تحول ملفاً يستخدم tabs ذات اربع مسافات الى ملفاً يستخدم tabs معيارية ذات ثمانية مسافات. يمكن للبرنامج ايضاً ان يحول الـ Tab الى مسافات، أو يستبدل المسافات بـ Tabs (وهي ايضاً تقنية ضغط ملف، رغم انها لا ينظر إليها كذلك). لاستخدام البرنامج، اختر الـ SpeedButtons التالية: Convert، أو Remove، أو Insert، ثم اضغط زر الـ Open لفتح ملف. اجب بـ Yes لبدء الملف. استخدم اوامر الـ Options وازرارها لتجهيز مواصفات البرنامج.

**تحذير:** تأكد من صنع بدائل لجميع الملفات قبل التشغيل مع الـ Tabs !  
ان بعض المجموعات من خيارات الـ tab يمكن ان تنتج تأثيرات غير مرغوبة، وقد تحتاج الى تجربة عدة مواصفات مختلفة لتحصل على النتيجة المنشودة. بعد تشغيل الملف، قد يكون من الممكن استعادة تهيئة النص الاصلى لهذا الملف، لذا تكون البدائل (أو النسخ الاحتياطية) هامة. لقد صممت البرنامج على حفظ الملفات الأصلية بعد إعادة تسميتها مع الامتداد bak، ولكنى اقترح بشدة ضرورة ان تحتفظ بنسخ احتياطية منفصلة من ملفاتك الأصلية بالإضافة الى ذلك التى ينشئها الـ Tabs.



**ملحوظة:** بالنسبة، لقد كتبت الـ Tabs اصلاً فى الـ Turbo Vision للـ DOS باستخدام الـ Turbo Pascal. ولقد استغرق تحويل البرنامج الى الـ Delphi بضع ساعات، وحوالى عشر دقائق لتحويله الى الـ Delphi 3 & 4 (كانت هناك حاجة الى قليل من التغير فى الـ Inidata unit بالإضافة الى تعديلات قليلة للعرض). ان كل برمجة واجهة التطبيق للمستخدم جديدة، ولكن الـ Functions وprocedures إنتاج الجداول لم تتغير الى حد كبير عن الـ code الأصلية.



### إنشاء الـ toolbar:

يحتوى الـ toolbar على الـ Panel object أو أكثر من الـ SpeedButtons. لإنشاء الـ toolbar إتبع هذه الخطوات:

- 1- أضف الـ Panel على الـ form. ان قيمة الـ Panel Height الافتراضية 41. إننى افضل 25، ولكن يمكنك استخدام أى قيمة تريدها.

٢- قم بتعيين Name مناسب للـ Panel مثل ToolbarPanel . هذا هو الأسم الذي يستخدمه الـ Tabs .

**ملحوظة:** بالرغم من ان Delphi يقدم الـ Toolbar الخاصة بال Win32 لإنشاء toolbars، الا ان هذا الفصل يوضح كيفية إنشاء الـ toolbar باستخدام انواع اخرى من الـ objects . ولان الـ ToolBars قد تم إدخالها بشكل مثالي في الـ Coolbar، فإننى سوف اشرح . الـ Toolbar مع المعلومات المذكورة عن الـ Coolbar في هذا الباب .

Note

٣- حدد خاصية الـ ToolbarPanel's Align بـ alTop أو alBottom، اعتماداً على ما اذا كنت تريد ان يظهر الـ toolbar فى أعلى أو اسفل النافذة . يمكنك ايضاً تصميم toolbar تظهر الى الحدود اليسرى (alLeft) أو اليمنى (alRight) بالرغم من ان المكان المعيارى هو فى أعلى الـ client area بالنافذة، تحت menu bar بالنافذة مباشرة .

٤- للحصول على toolbar ذى صفين أو أكثر من الـ SpeedButtons، أضيف panels متعددة على الـ form وحدد خاصية الـ Align لهم بـ alTop . هذا يرتب الـ panels كالأجهزة الموضوعة تحت الـ countertop . يمكنك عندئذ إضافة SpeedButtons على الـ Panel .

٥- بعد إنشاء الـ Panel، قم بحذف الـ Caption الخاص بها وادخل ماتحتاج من الـ SpeedButton . قم دائماً بإنشاء الـ Panel أولاً، ثم اسقط الـ SpeedButtons عليها- هذا يجعل Delphi يعرف ان الـ Panel هى صاحبة الـ SpeedButton . إننى اجعل اسم الـ SpeedButton الخاصة بـ OpenSB و ConfigSB، والذي يذكرنى بانواع الـ component types . فى الغالب إنك تود ان تنشئ الـ OnClick لكل الأزرار، ويمكنك التشارك فى الـ event handlers مع اوامر وازرار القائمة .

هذا هو كل ما تحتاج فعله لإنشاء toolbar من الـ SpeedButtons . بسيط، أليس كذلك؟ .

ولكن، هناك المزيد عن التحكم فى toolbar أكثر مما تراه العين . على سبيل المثال، إنك غالباً تريد ان تضيف نص الـ Hint لكل زر . لتفعل هذا، اكتب وصفاً

## الباب السابع : إنشاء الـ Toolbars ، الـ Coolbars ، والـ Status Panels

مختصراً في خاصية الـ Hint للـ SpeedButton's ، وحدد الـ ShowHint بـ True . يظهر عندئذ مربع معلومات صغير عندما يوضح مؤشر الفأرة على الزر لثانية واحدة . هناك سبب بسيط لعدم إضافة ملحوظات لأزرار الـ toolbar الخاصة بك - إنها تجعل البرامج ايسر في الاستخدام .

تستطيع الـ Panels أيضاً أن تستجيب للـ events ، فمثلاً ، يمكنك إنشاء الـ Panel object لـ OnClick لأداء عمل عندما يضغط المستخدم ، ليس على زر ، ولكن داخل الـ Panel نفسها . ومؤخراً في هذا الباب ، سوف اشرح كيفية استخدام هذه الخاصية لإنشاء floating toolbar يمكن للمستخدمين ضغطه وسحبه حول النافذة . وبوجه عام ، من الأفضل عدم توجيه الـ Panels events - فإن المستخدمين يعرفون بديهياً أنه يمكنهم ضغط أزرار في toolbar ، ولكن توفير toolbar قابل للضغط قد يكون أكثر تضليلاً عن المساعدة . (ان الـ Coolbar يعد خياراً أفضل اذا كنت تريد توفير هذا المستوى من التفاعل في واجهة تطبيق المستخدم الخاص بك . انظر الـ Coolbars في هذا الباب لمزيد من المعلومات حول إنشاء toolbar قابلة للتشكيل في وقت التشغيل) .

### الـ SpeedButtons

عند تصميم toolbar ، ضع في الاعتبار اذا كنت تريد string-loaded Buttons بمجرد ان تطلق زر الفأرة ، أو مجموعة من الـ objects تعمل مثل الـ radio buttons . على سبيل المثال ، يمكنك إنشاء SpeedButton "sticky" تبقى مضغوطة عند دفعها ؛ يمكنك أيضاً إنشاء مجموعات SpeedButton بحيث ان ضغط زر واحد يؤدي الى إبطال إختيار آخر .

ان الـ SpeedButton المحملة هي الاسهل في إنشاءها . فقط أضف الـ SpeedButton object في الـ Panel لـ toolbar ، وقم بإنشاء الـ OnClick الخاص بها . (انظر الباب السابق لمزيد من المعلومات عن برمجة الـ SpeedButtons ، وكيفية التعامل مع الـ glyph Bitmap ، وإظهار صور الزر) . تعتبر spring loaded Buttons نافعة في عمليات مثل فتح ملف ، طباعة ، ومهام أخرى تبدأ مباشرة . بالتبادل ، يمكنك إنشاء SpeedButtons on/off تبقى الى أسهل عندما تضغطها ، ثم تقفز عندما تضغطها مرة أخرى .

## On/off (sticky) SpeedButtons

ان هذا النوع من ال Toolbar Button يعتبر نافعا للخيارات التى تختار خصائص البرنامج . فمثلاً ، فى برنامج معالج كلمات قد يختار ال SpeedButton أنماط كتابة مثل الخط السميك ، المنحرف ، وضع الخط تحت كلمات . قد يضغط المستخدمون الزر مرة واحدة لتحويل نص كتابة تم إختياره الى الخط السميك ؛ ثم يضغطوا الزر مرة أخرى لإعادة النص الى الوضع الطبيعى .

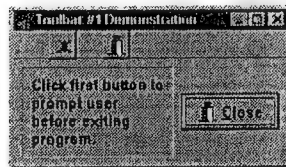
إتبع هذه الخطوات لإنشاء SpeedButtons :

١ - حدد خاصية ال GroupIndex لل SpeedButtons بأية قيمة غير صفر لا يستخدمها أى control آخر فى toolbar .

٢ - حدد خاصية ال AllowAllUp لل SpeedButton's بـ True .

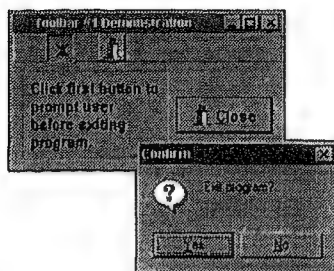
٣ - باختيارات حدد خاصية ال Down لل SpeedButton بـ True لعرض الزر أولاً فى حالة ضغطه .

يوضح التطبيق ال spring-loaded و sticky SpeedButtons ، وتوجد ملفات البرنامج على القرص المدمج فى دليل ال Source\Toolbar1 . يوضح شكل (٧-٢) مظهر البرنامج . ان زر ال SpeedButton الـ sticky ؛ إنه يظل الى اسفل عندما تضغطه . افعل هذا الآن واختر Close ، الذى يحثك على إعطاء تأكيد قبل إنهاء البرنامج ، كما هو موضح فى شكل (٧-٣) . اطلق زر ال SpeedButton الـ sticky لتخرج بشكل طبيعى دون عرض message dialog .  
توضح القائمة (٧-١) ال source code للبرنامج .



شكل (٧-٢) : عرض برنامج ال Toolbar1 يستخدم إثنين من ازرار ال SpeedButtons فى toolbar بسيط. الزر الـ sticky أى يبقى الى اسفل عندما تضغطه

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels



شكل (٧-٣): عندما الـ SpeedButtons الواقع في أقصى اليسار الخاص بالـ toolbar مضغوطاً، يطلب الـ Toolbar1 تأكيداً قبل الإنهاء

القائمة (٧-١): Toolbar1\Main.pas:

```
unit Main;
```

```
interface
```

```
uses
```

```
SysUtils, Windows, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;
```

```
type
```

```
TMainForm = class(TForm)
```

```
    Panel1: TPanel;
```

```
    PromptSB: TSpeedButton;
```

```
    ExitSB: TSpeedButton;
```

```
    CloseBitBtn: TBitBtn;
```

```
    Label1: TLabel;
```

```
    Bevel1: TBevel;
```

```
    procedure ExitSBClick(Sender: TObject);
```

```
    procedure FormActivate(Sender: TObject);
```

```
private
```

```
    { Private declarations }
```

```
public
```

```

{ Public declarations }
end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.ExitSBClick(Sender: TObject);
begin
    if not PromptSB.Down then
        Close
        else if MessageDlg('Exit program?', mtConfirmation,
            [mbYes, mbNo], 0) = mrYes then
            Close;
end;

procedure TMainForm.FormActivate(Sender: TObject);
begin
    ExitSB.Glyph := CloseBitBtn.Glyph;
end;

end.

```

يوضح ملف الـ Main.pas الخاص بالـ Toolbar1 إثنين من التقنيات الهامة للعمل بالـ SpeedButtons. للزر الـ sticky يمكنك استخدام عبارة if لاستكشاف ما إذا كان الزر مضغوطاً. على سبيل المثال، انظر السطر الأول في procedure ExitSBClick، والذي ينفذ العبارة إذا لم يكن زر الـ PromptSB مضغوطاً:

if not promptSB.Down then ...

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

ثانياً، توضح الـ module كيفية سرقة glyph من object آخر للعرض في واجهة الـ SpeedButton. لاحظ في شكل (٧-٢) ان الـ SpeedButton على اليمين وزر الـ Close لهما نفس الـ glyph. يسرق الـ SpeedButton هذه الصورة في وقت التشغيل بتنفيذ العبارة التالية في الـ OnCreate أو الـ OnActivate:

```
ExitSB.Glyph := CloseBtn.Glyph;
```

### مجموعات الـ SpeedButton:

ان إنشاء مجموعة من الـ SpeedButton اللاصقة والتي تعمل مثل radio button يعتبر procedure أساسى. فقط اختر كل الازرار فى المجموعة وحدد خصائص الـ GroupIndex لهم بأى قيمة غير الصفر لم يتم استخدامها من جانب أى toolbar control آخر. ومع هذا التحديد، فإن ضغط زر لم يتم اختياره يؤدي الى ظهور الزر الذى يتم اختياره حالياً مرة اخرى.

فى البداية، يمكن ان تكون جميع الازرار فى حالة الرفع. ولكن بمجرد ان يضغط المستخدم زراً، فعلى الأقل يبقى زراً واحداً الى اسفل. اذا كنت تريد ان يتمكن المستخدمون من إبطال اختيار جميع الازرار فى مجموعة، حدد خاصية الـ AllowAllUp بـ True لكل زر يمكن ان يعود الى حالة الرفع.

فى وقت التصميم، يمكنك تحديد خاصية الـ Down للـ SpeedButton بـ True لتعرضها فى البداية فى حالة إنخفاض. ولكن لكى يعمل هذا، يجب ان تحدد خاصية الـ GroupIndex للزر بأى قيمة غير الصفر. استخدم قيمة فريدة بين كل الازرار الا اذا كنت تنوى تجميعهم كما تفعل فى مجموعة الـ radio buttons.

**Tip فكرة:** اذا رفضت خاصية الـ Down للـ SpeedButton ان تتغير الى True، حدد خاصية الـ GroupIndex للزر بأى قيمة غير مستخدمة غير الصفر. اذا ظل الـ SpeedButton ملتصقاً فى الموضع السفلى - أى اذا رفضت خاصية الـ Down ان تعود الى الـ False، حدد الـ AllowAllUp بـ True. يمكنك عندئذ إعادة تحديد الـ Down ليقفز الزر.

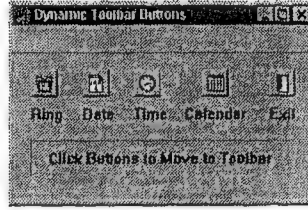
يظهر برنامج الـ Tabs [انظر شكل (٧-١)] كيفية برمجة مجموعة من الـ SpeedButtons لكى تعمل كـ toolbar قم بتشغيل الـ Tabs واختار ازرار الـ



Insert، Remove، والـ Convert لاختيار العملية التى تريد أداؤها على ملف النص. واحداً من هذه الأزرار الثلاث يكون دائماً فى حالة إنخفاض ويعمل الزران الآخران فى الـ Tabs كـ spring-loaded push buttons.

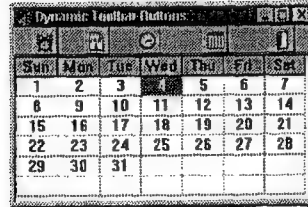
## Dynamic toolbars

باستخدام مجموعة من الـ MouseDown والـ MouseUp والـ OnClick events، يمكنك انشاء dynamic toolbars يمكن للمستخدمين تشكيلها فى وقت التشكيل. والتطبيق (Toolbar2) الموضح فى شكل (٧-٤)، يظهر هذه التقنيات، ويوضح أيضاً بعض طرق استغلال الـ component objects و الـ event handlers باستخدام عبارات برنامج. توجد ملفات البرنامج على القرص المدمج فى دليل Source\Toolbar2.



شكل (٧-٤): اضغط الـ SpeedButtons على form الـ Toolbar2 لتحويلها الى Toolbar يمكنك عندئذ ضغط الأزرار لى تقوم بأداء اعمالها

قم بتشغيل البرنامج واضغط الـ SpeedButton فى الـ form لنقلها الى داخل الـ toolbar بعد نقل زر الى toolbar اضغط الزر لأداء عمل مثل عرض تقويم [انظر شكل (٧-٥)]. اضغط زر الـ calendar مرة اخرى لإزالة التقويم. يجب ان تنقل الـ SpeedButton الى Exit الى toolbar قبل ان تضغطه لإنهاء البرنامج. توضح القائمة (٧-٢) الـ source code للبرنامج.



شكل (٧-٥): بعد نقل الـ SpeedButtons الخاصة بالـ Toolbar2 الى Toolbar، اضغط زر الـ calendar لعرض تقويم شهرى

## الباب السابع : إنشاء الـ Toolbars ، الـ Coolbars ، والـ Status Panels

القائمة (٧-٢) : Toolbar2\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Buttons,
  Grids,
  Calendar;

type
  TMainForm = class(TForm)
    ToolbarPanel: TPanel;
    RingSB: TSpeedButton;
    DateSB: TSpeedButton;
    TimeSB: TSpeedButton;
    CalendarSB: TSpeedButton;
    ExitSB: TSpeedButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Bevel1: TBevel;
    Calendar1: TCalendar;
    procedure SBMouseDown(Sender: TObject;
      Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure SBMouseUp(Sender: TObject; Button:
      TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure RingSBClick(Sender: TObject);
    procedure DateSBClick(Sender: TObject);
    procedure TimeSBClick(Sender: TObject);
    procedure CalendarSBClick(Sender: TObject);
    procedure ExitSBClick(Sender: TObject);
```

```

private
    { Private declarations }
    function InToolbar(Sender: TObject): Boolean;
    public
    { Public declarations }
    end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

const
    isNotInToolbar = 0;    { SpeedButton Tag flag }
    isInToolbar    = 1;    { SpeedButton Tag flag }

{- Returns True if Sender is in the toolbar }
function TMainForm.InToolbar(Sender: TObject): Boolean;
begin
    with Sender as TSpeedButton do
        if Tag = isNotInToolbar then
            begin
                Tag := isInToolbar; { Set Tag flag }
                Result := False;     { Return function result = False }
            end else
                Result := True;      { Return function result = True }
    end;

{- Assign OnClick event handlers for buttons not in toolbar }
procedure TMainForm.SBMouseDown(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    with Sender as TSpeedButton do
        if Tag = isNotInToolbar then

```

## الباب السابع : إنشاء الـ Toolbars , الـ Coolbars , والـ Status Panels

////////////////////////////////////

```
begin { Assign OnClick event handler to a button }
  if Sender = RingSB then
    RingSB.OnClick := RingSBClick
    else if Sender = DateSB then
      DateSB.OnClick := DateSBClick
    else if Sender = TimeSB then
      TimeSB.OnClick := TimeSBClick
    else if Sender = CalendarSB then
      CalendarSB.OnClick := CalendarSBClick
  else if Sender = ExitSB then
    ExitSB.OnClick := ExitSBClick;
  end;
end;

{- Move buttons into the toolbar }
procedure TMainForm.SBMouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  with Sender as TSpeedButton do
    if Tag = IsNotInToolbar then
      begin { Move button into the toolbar }
        Parent := ToolbarPanel; { ToolBar now owns the button }
        Top := 0;               { Reposition button }
      end;
end;

{- Respond to Ring button click }
procedure TMainForm.RingSBClick(Sender: TObject);
begin
  if InToolbar(Sender) then
    MessageBeep(0);
end;

{- Respond to Date button click }
procedure TMainForm.DateSBClick(Sender: TObject);
begin
```

```

if InToolbar(Sender) then
    ShowMessage('The date is ' + DateToStr(Date));
end;

{ - Respond to Time button click }
procedure TMainForm.TimeSBClick(Sender: TObject);
begin
    if InToolbar(Sender) then
        ShowMessage('The time is ' + TimeToStr(Time));
end;

{ - Respond to Calendar button click }
procedure TMainForm.CalendarSBClick(Sender: TObject);
begin
    if InToolbar(Sender) then
        with Calendar1 do
            begin
                Align := alClient;
                Visible := not Visible;
            end;
end;

{ - Respond to Exit button click }
procedure TMainForm.ExitSBClick(Sender: TObject);
begin
    if InToolbar(Sender) then
        Close;
end;

end.

```

لإنشاء Toolbar2، قم بأداء الخطوات التالية :

- ۱- أضف Panel وعدد من ال objects SpeedButton على ال form.
- ۲- قم بإنشاء OnClick event handlers مثل RingSBClick و DateSBClick لكل زر.

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

٣- فى الـ Object Inspector، قم بابرار كل الـ OnClick event واضغط Del لحذفه. هذا يترك الـ procedure فى الـ source module، ولكن يفصله عن الزر- وهذا ضرورياً لانه فى المرة الأولى التى يقوم فيها المستخدمون بضغط الزر ينقله البرنامج الى الـ toolbar بدلاً من ان يؤدي عملاً.

للإشارة اذا ما كان هناك زرأ فى الـ toolbar يفحص البرنامج حقل الـ Tag الخاص بالزر، والذي يمكن ان يساوى واحد من الثابتين، وهما الـ IsNotInToolbar أو الـ isInToolbar. يعرف البرنامج هذه الثوابت مباشرة عقب كلمة الـ implementation فى الـ unit module. وتعود الـ InToolbar Function بـ True أو False اعتماداً على قيمة الـ Tag وتغير هذه القيمة الى الـ isInToolbar.

**Tip** فكرة: يمكنك استخدام حقل الـ Tag الخاص بالـ object لأى قيمة عدد صحيح. كما هو موضح هنا، ان الاستخدام الأمثل هو عبارة عن flag يشير إلى بعض الحقائق عن حالة الـ object..

ان كلاً من الـ SpeedButtons يتشارك فى نفس الـ MouseDown و MouseUp event handlers. فى الـ SBMouseDown procedure، تتحقق عبارة if اذا ما كان الـ Tag مساو لـ IsNotInToolbar، وفى هذه الحالة يمكننا ان نفترض ان المستخدم قد ضغط زرأ ينقله الى الـ toolbar. لضمان ان الاستخدام التالى لهذا الزر ينفذ عملاً، يعيد الـ SBMouseDown إلحاق الـ OnClick الخاص بالزر، الذى كان قد تم فصله سابقاً.

عند إطلاق الفأرة، يتحقق الـ SBMouseUp procedure مرة اخرى عما اذا كان الـ Tag مساوياً لـ IsNotInToolbar. فإذا كان كذلك، ينقل الـ procedure الزر الى الـ toolbar بتحديد خاصية الـ Top للزر بـ (0)، والتى تعيد وضع الزر بالنسبة لحد الـ Panel. وتؤدي هاتان العبارتان الاعمال الضرورية:

```
Parent := ToolbarPanel;
```

```
Top := 0;
```

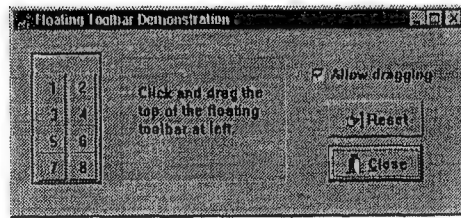
ويقوم كل الـ OnClick event باستدعاء الـ InToolbar function لفحص ما اذا كان الزر قد تم نقله الى شريط. اذا عادت الـ InToolbar بـ True، يؤدي الـ

event handler عمل الزر. وبالنسبة، انظر الى كيف يقوم ال  
CalendarSBClick procedure باظهار وأخفاء ال calendar object ببساطة  
بربط خاصية ال Visible له بهذه العبارة:

Visible := not Visible;

### ،Floating toolbars

من الممكن ان لا تظل ال Toolbar panels ثابتة فى مكانها. يمكنك ايضاً  
إنشاء toolbars للمستخدمين ضغطها وسحبها الى أى مكان على ال form. ان  
التطبيق Toolbar3، الموضح فى شكل (٦-٧)، يكشف هذه التقنية المفيدة، والتي  
تعتبر قيمة بشكل خاص فى البرامج ذات العروض المزدحمة، أو أولئك الذين  
يعرضون صور جرافيكية كبيرة. وتوجد ملفات البرنامج على القرص المدمج فى  
دليل ال Source\Toolbar3. يستطيع المستخدمون ببساطة سحب floating  
toolbar بعيداً عن الشاشة بدلاً من الاضطرار الى تحريك النافذة. لتجربة هذا  
البرنامج، اضغط واسحب الحد العلوى لل toolbar. اختر أى زر، والذي يؤكد  
ببساطة اختيارك. إغلق ال check box لتثبيت ال toolbar فى مكانه. اضغط  
Reset لإعادة ال toolbar الى مكان البداية. وتوضح ال source code فى القائمة  
(٣-٧) تفاصيل إنشاء ال Panel القابل للسحب. بعد القائمة، سوف اشرح كيف  
يعمل البرنامج.



شكل (٦-٧): يوضح ال Toolbar3 كيفية  
إنشاء ال floating toolbar لل SpeedButtons.

**ملحوظة:** يقدم Delphi 4 الان نوافذ قابلة للوصل، والتي تبسط الى  
حد كبير التقنيات الموجودة فى هذا الفصل. ولكن لا يزال بإمكانك  
استخدام هذه الاساليب اذا كنت تستخدم النسخ الأولى من Delphi،



## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

ولكن انظر "Creating Docking Controls" في الباب الثاني عشر لمعرفة طريقة أخرى أنظر لإنشاء dockable windows يمكن استخدامها كـ floating toolbars.

القائمة (٧-٣): Toolbar3\Main.pas

unit Main;

interface

uses

SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, Buttons, ExtCtrls, StdCtrls;

type

TMainForm = class(TForm)

FloatingToolbar: TPanel;

SpeedButton1: TSpeedButton;

SpeedButton2: TSpeedButton;

SpeedButton3: TSpeedButton;

SpeedButton4: TSpeedButton;

SpeedButton5: TSpeedButton;

SpeedButton6: TSpeedButton;

SpeedButton7: TSpeedButton;

SpeedButton8: TSpeedButton;

BitBtn1: TBitBtn;

Label1: TLabel;

Bevel1: TBevel;

AllowDraggingCB: TCheckBox;

ResetBitBtn: TBitBtn;

procedure FormCreate(Sender: TObject);

procedure SpeedButton1Click(Sender: TObject);

procedure FloatingToolbarMouseDown(Sender:  
TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

procedure FormMouseUp(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y:



```

Integer);
procedure TFormMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
  procedure ResetBitBtnClick(Sender: TObject);
  procedure AllowDraggingCBClick(Sender: TObject);
private
{ Private declarations }
  Dragging: Boolean;
  XOffset, YOffset: Integer;
  procedure MoveToolbar(X, Y: Integer);
public
{ Public declarations }
end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

{- Initialize }
procedure TMainForm.FormCreate(Sender: TObject);
begin
  Dragging := False;
end;

{- Display number of selected SpeedButton in toolbar }
procedure TMainForm.SpeedButton1Click(Sender: TObject);
begin
  ShowMessage('You selected button number ' +
    IntToStr(TSpeedButton(Sender).Tag));
end;

{- Start dragging operation on clicking in toolbar }
procedure TMainForm.FloatingToolbarMouseDown(Sender: TObject);

```

## الباب السابع : إنشاء الـ Toolbars ، الـ Coolbars ، والـ Status Panels

```
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    if AllowDraggingCB.Checked then
    begin
        Dragging := True; { Dragging operation in effect }
        SetCapture(Handle); { Send all mouse messages to form }
        XOffset := X; { Save mouse coordinates to compute }
        YOffset := Y; { offset from top-left corner }
    end;
end;
```

```
{ - End dragging operation on releasing mouse button }
procedure TMainForm.FormMouseUp(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    if Dragging then { Ignore if not dragging }
    begin
        MoveToolbar(X, Y); { Move toolbar to final location }
        Dragging := False; { End dragging operation }
        ReleaseCapture; { Return message handling to normal }
    end;
end;
```

```
{ - Move toolbar if dragging operation in progress }
procedure TMainForm.FormMouseMove(Sender: TObject; Shift:
    TShiftState; X, Y: Integer);
begin
    if Dragging then { Ignore if not dragging }
    MoveToolbar(X, Y); { Move toolbar to mouse location }
end;
```

```
{ - Move the toolbar to mouse location X and Y }
procedure TMainForm.MoveToolbar(X, Y: Integer);
begin
    FloatingToolbar.Left := X - XOffset; // Adjust location of
```

```

FloatingToolbar.Top := Y - YOffset; // panel top-left corner
end;

{- Reset toolbar to startup location }
procedure TMainForm.ResetBitBtnClick(Sender: TObject);
begin
  with FloatingToolbar do
    begin
      Left := 24;
      Top := 24;
    end;
    AllowDraggingCB.Checked := True;
end;

procedure TMainForm.AllowDraggingCBClick(Sender: TObject);
begin
  with Label1 do
    Enabled := not Enabled;
  end;

end

```

لإنشاء floating toolbar، أضف الـ Panel على الـ form وعين له خاصية Name مناسبة. فى الـ Toolbar3، قمت باستخدام اسم FloatingToolbar، ولقد غيرت عرض الـ object الى 50- بالتحديد ضعف عرض الـ SpeedButton. ان الـ Height الخاص بالـ toolbar ليس على درجة من الاهمية. اجعله طويلاً بما يكفى لان يحمل ازرارك بالإضافة الى مسافة بسيطة عند القمة. يمكن للمستخدمين ان يضغطوا ويسحبوا داخل هذه المساحة لنقل toolbar إلى مكان آخر.

ان خصائص الـ BevelOuter والـ BevelInner الخاصة بالـ toolbar كلاهما محدد بـ bvRaised، ولكن قد تريد ان تختبر انماط وتحديدات اخرى للـ Panel. ان الـ SpeedButtons الموجودة فى الـ Panel تعتبر نسخة معيارية. ببساطة أضف قدر ما تحتاج من الازرار.

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

للتحكم في السحب فإنك تحتاج الى ثلاث متغيرات، وهى مقرر فى قطاع الـ private للـ TMainForm. وهذه المتغيرات هى :

● **Dragging** : وهو متغير Boolean والذي، عندما يكون بـ True، يشير الى ان عملية سحب يتم تنفيذها.

● **Xoffset** : هو عدد الـ pixels الموجودة بين الحافة اليسرى للـ Panel الخاصة بـ toolbar والأحداثى X التابع للفأرة. وهذا يجعل السحب اكثر واقعية بالسماح للمستخدمين ان يضغطوا الفأرة فى أى مكان على visible surface.

● **Yoffset** : عدد الـ pixels الواقعة فما بين قمة الـ Panel الخاصة بـ toolbars والإحداثى Y- التابع للفأرة.

بالإضافة الى هذه المتغيرات، الـ class تعرف procedure خاصاً واحداً، وهو MoveToolbar. قم بتمرير إحداثيات X- و Y- الخاصة بمكان الفأرة (من خلال الـ MouseMove event، مثلاً) لتحريك الـ toolbar الى هذا المكان.

اكتب OnCreate event handler للـ form وحدد متغير الـ Dragging بـ False. هذا الذى نحتاجه عند البداية.

بعد ذلك، قم بإنشاء الـ OnMouseDown لـ FloatingToolbar. تأكد من ان تفعل هذا للـ Panel، وليس للـ form لتحقق Toolbar3 مما اذا كان الـ Allow dragging check box بـ on - واذا لم يكن كذلك، ينتهى الـ procedure، مما يمنع بشكل فعال الـ toolbar من الانتقال.

هناك اربعة عبارات يبدئن عملية السحب (راجع الـ MouseDown فى القائمة):

```
if AllowDraggingCB.Checked then
begin
    Dragging := True;
    SetCapture(Handle);
    XOffset := X;
    YOffset := Y;
end;
```

أولاً، يحدد البرنامج ال Dragging بـ True بحيث يمكن لل procedures الأخرى ان تكتشف ان هناك عملية سحب تتم. وال SetCapture، وهي API Windows function، ترسل كل رسائل الفأرة الى ال active window وبذلك، فى اثناء السحب، لا يستطيع المستخدمون استخدام الفأرة للتحويل الى مهام اخرى. اخيراً، هناك تعيينين يبدءان ال XOffset وال YOffset لإحداثيات الفأرة، والذان متعلقان بال Panel الذى يحدث له ال MouseDown event.

عندما تطلق زر الفأرة، يتولى الأمر ال MouseUp event الخاص بال form. لا تتم بإنشاء هذا ال event لل Panel- يجب عليك ان تنشئ ال MouseUp لل form، بالرغم من ان عملية السحب بدأت لل Panel. فى الواقع، عند إنشاء أى نوع من ال objects القابلة للسحب، يجب ان يكون ال MouseUp event لل form، بالرغم من الحقيقية ان ال MouseDown يكون لل object. والسبب فى هذا هو ان ال SetCapture، والذى يتم استدعاءه فى ال MouseDown، يؤدى الى جعل كل رسائل الفأرة التى يتم إرسالها الى ال active window وهذه النافذة هي ال form- وليس ال Panel- ولذلك، اثناء ال SetCapture، لا تتلقى ال Panel ال MouseUp وال MouseMove.

ويقوم ال MouseUp event أولاً بالتحقق مما اذا كان ال Dragging محدد بـ True. اذا كان كذلك، فإن ال MoveToolbar ينقل ال toolbar الى موضعه النهائى عند احداثيات الفأرة x و y اللذان تم تعديلهما لتشغيل اللوحة. ويحدد ال procedure بـ False، مما يلغى عملية السحب، ويستدعى ال ReleaseCapture لإعادة رسائل الفأرة الى انماط المرور الطبيعية لها.

**فكرة:** دائماً اجعل استدعاء ال SetCapture مرتبط بال ReleaseCapture أو سيفقد مستخدمى برنامجك التحكم فى الفأرة. إذا حدث هذا، قد تفقد عملائك.

والخطوة الأخيرة فى إنشاء ال floating toolbar هي ال MouseMove event. الخاص بال form. مرة اخرى، بسبب استدعاء ال SetCapture، يجب ان تنشئ هذا ال handler لل form event، وليس ال toolbar. ويتحقق ال procedure مما اذا كان ال Dragging محدد بـ True، مما يشير الى ان عملية

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

الضغط والسحب تتم الآن، وإذا كان الأمر كذلك، فإنه يستدع الـ MoveToolbar لتحريك الـ Panel الخاص بالـ toolbar الى مكان الفأرة. ولأن الـ toolbar يمتلك الـ SpeedButton objects الخاص به، فإن هذه الـ objects تتبعه ألياً- فلا يجب عليك إعادة تحديد موضعها؛ عليك فقط ان تنقل الـ Panel الخاص بالـ toolbar.

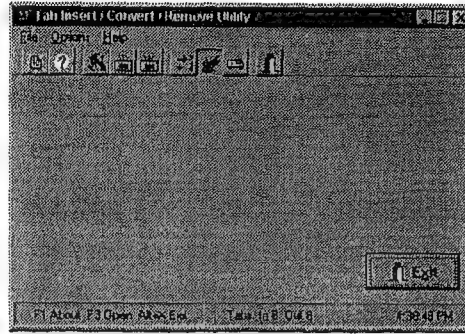
والـ procedure الأخرى في الـ Toolbar3 لها اهداف واضحة. يقوم الـ MoveToolbar بإعادة وضع الـ FloatingToolbar بتحديد قيم جديدة لخصائص الـ Left والـ Top الخاصة به. وي طرح قيم تشغيل الفأرة من الـ X والـ Y يسمح للمستخدمين ان يضغطوا الفأرة في أى مكان من الـ visible surface للـ Panel. (جرب تحديد الـ X والـ Y دون طرح قيم التشغيل لترى ضرورة هذه الخطوة).

والـ ResetBitBtnClick، الذى يتم استدعاه عندما تضغط زر الـ Reset، يعيد الـ floating toolbar الى مكان بدايته. من الصواب دائماً ان يكون لديك هذه الإمكانية لربما يفقد المستخدمون أثر الـ toolbar الأمر الذى قد يحدث بسهولة عند إعادة تحديد حجم نافذة البرنامج.

## •Status Panels

تشبه الـ status panels الـ toolbars، ولكنها غالباً ما تعرض نصاً بدلاً من الـ SpeedButton objects. وتظهر الـ status panels دائماً فى اسفل النافذة، وقد يكون لها قطاعات فرعية تجعل قرائتها ايسر. على سبيل المثال، Microsoft Word، الذى استخدمته لكتابة هذا الكتاب، يعرض رقم الصفحة الحالية، قيم السطر والعمود، ومؤشرات لوحة المفاتيح فى الـ Status panels مقسمة. ومن حسن الحظ انها لم تعد تعرض الوقت، والذى يذكرنى دائماً بتأخيرى فى إنهاء هذه الكتب شكل (٧-٧) يعرض الـ status panels من خلال برنامج الـ Tabs. وتوضح لوحة المفاتيح الـ functions والـ Alt keys، ومواصفات الـ input and output الـ tab-width، والوقت الحالى. ان كل قطاع فرعى فى اللوحة يعتبر فى حد ذاته الـ Panel object منفصل.

يمكنك أيضاً عرض الـ online help أو رسائل خطأ فى الـ status panel. ان Delphi قد جعله من السهل إنشاء الـ status panels.



شكل (٧-٧)، برنامج الـ Tabs يعرض sectional status panel في اسفل حد النافذة

**ملحوظة:** بالرغم من ان Delphi يقدم الـ StatusBar component الخاص بالـ Win32 لإنشاء status panels، فإن هذا الفصل يوضح كيفية إنشاء status panels باستخدام انواع اخرى من الـ objects. يشرح الفصل الاخير فى هذا الباب كيفية استخدام الـ StatusBar component.



### إنشاء status panel:

لإنشاء status panel، إتبع هذه الخطوات:

- ١- أضف الـ Panel على الـ form لتكون القاعدة لكل panels فرعى.
- ٢- إعط Name لهذه الـ Panel الرئيسية مثل StatusPanel، وقم بتغيير خاصية الـ Height لها لتصبح 25. يمكنك استخدام قيمة اصغر، ولكن تذكر، ليس كل مستخدمى الحاسوب ليس لديهم ابصار حادة، ٦ / ٦، فلا تجعل لوحاتك صغيرة جداً.
- ٣- للحصول على تأثير 3D افضل، حدد خاصية الـ BevelInner للـ Panel بـ bvRaised، وحدد الـ BevelOuter بـ (القيم الافتراضية).
- ٤- حدد الـ BorderWidth بـ (0) (القيمة الافتراضية).
- ٥- احذف الـ Caption.
- ٦- لتوصيل الـ Panel الى حافة زر النافذة، قم بتغيير الـ Align الى alBottom.

## الباب السابع : إنشاء ال Toolbars، ال Coolbars، وال Status Panels

بالطبع يمكنك فى إختيار انماط اخرى وفى وضع ال Panel فى أى مكان تريده .

**فكرة:** كثير من برامج ال Windows الناجحة تستخدم بنط ال MS Sans Serif ذى الثمانى نقاط لل status panel's text . مع هذا البنط ، يمكنك تعبئة حوالى ١٠٠ رمز فى status panel's خاصة بنافذة مكبرة على عرض ٦٤٠×٤٨٠ معيارى . ويبدو هذا البنط جيداً ايضاً مع ال resolutions الأعلى .

### تقسيم ال status panel الى قطاعات:

لتقسيم status panel الى قطاعات ، أضف ال panel الإضافية فى ال status panel ( ال Panel object الرئيسى ) اختر اسماء يسهل تذكرها لكل Panel فرعية . على سبيل المثال ، يسمى ال Tabs ال Panels الفرعية الخاصة به بـ KeyPanel ، TabPanel ، و TimePanel . استخدم نفس تحديدات ال 3D لل Panel الرئيسية ، ولكن حدد ال BevelOuter بـ None . لمحاذاة ال Panels الفرعية تلقائياً ، قم بتغيير خصائص ال Align لها لتصبح alLeft أو alRight .

عندما تحدد ال Align بـ alLeft أو alRight ، تشترك كل panel فرعية تلقائياً مع أى panel اخرى فى status panel الرئيسى . بعد تحديد ال Align مبدئياً ، أعد تحديد هذه الخاصية بـ alNone . يمكنك حينئذ ضغط وسحب ال panels الفرعية الى مكان آخر - لزيادة كمية المسافة بين القطاعات ، مثلاً . هذه هى الطريقة التى انشأت بها المسافات المرتفعة بين ال panels الفرعية فى تطبيق ال Tabs [انظر شكل (٧-٧) ] .

يقدم الجدول (٧-١) خصائص هامة لل panel objects الفرعية وال StatusPanel فى تطبيق ال Tabs . قد تجد هذه المعلومات مفيدة فى بناء ال status panels المقسمة الخاصة بك .

### تحديث status panels:

لتغيير النص فى ال status panel ، ببساطة قم بتحديد خصائص ال Caption . على سبيل المثال ، لعرض الوقت ، نفذ Tabs هذه العبارة فى ال



OnTimer الخاص بالTimer object :

TimePanel.Caption := TimeToStr(Time);

جدول (١-٧)؛ خصائص Panel Object Properties تطبيق Tabs

Component	Name	Property	Value
Panel	StatusPanel	Align	alBottom
		BevelInner	bvLowered
		BevelOuter	bvRaised
		Height	25
Panel	KeyPanel	Align	alLeft
		BevelInner	bvLowered
		BevelOuter	bvNone
		Font:Name	MS Sans Serif
		Font:Size	8
Panel	TabPanel	Align	alNone
Panel	TimePanel	Align	alRight

هذا يؤدي الى ان يتقدم الوقت تلقائياً. ان ال Interval الخاص بال Timer محدد بـ ١٠٠٠ مللى ثانية (أى ثانية واحدة). يمكنك ايضاً تحديد Caption strings لعرض عناصر النص الثابت فى وقت التشغيل ، أو يمكنك كتابة strings فى خصائص ال Caption باستخدام ال Object Inspector .

رغم ذلك ، عليك تجنب الحاجة الى الزج بالرموز الفردية فى لوحة ال Captions الفرعية. على سبيل المثال ، من العبث ان تدخل ارقام للساعة ، والدقيقة ، والثانية فى ال Caption الخاص بال TimePanel- وهى محاولة اضاع فيها الكثير من واضعى البرامج فى ال Delphi الساعات لبرمجتها . ومن الأسهل ، والأكثر فعالية فى الغالب ، هو ان تعيد تحديد ال Caption strings بأكمله كل ثانية ، بالرغم من ان هذا قد يبدو مضيعاً للوقت .

قد تجد انه من الضروري ، فى بعض الأحيان ، ان تعرض نصاً فوق التقسيمات الفرعية لل status panel's على سبيل المثال ، يمكنك عرض ملحوظة online أو رسالة خطأ . بشكل مؤقت ، إنك تريد ان تكتب فوق ال panel's الفرعية

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

لتعرض الرسالة، ثم، عندما يضغط المستخدم مفتاح أو يستخدم الفأرة، يمكنك استعادة الـ status panel's الى طبيعتها.

وهذه التقنية يسهل تنفيذها. حدد الـ Alignment الخاص بالـ panel's الرئيسية بـ taLeftJustify، وقم بإخفاء الـ panel's الفرعية عندما تريد عرض الرسالة. قم بتحديد strings لخاصية الـ Caption للوحة الرئيسية. (أخيراً، اعكس هذه الخطوات لإعادة الـ status panel الى طبيعتها.

على سبيل المثال، يعرض برنامج الـ Tabs عبارة خطأ باستخدام code شبيهة بالآتي. أولاً، يقوم البرنامج بإخفاء الـ panel's الفرعيتين اللتان تعرضان مواصفات الـ the tab-width و function key labels (ان الـ panel time الفرعية ليست عقبة، ويمكن ان تبقى مرئية):

```
KeyPanel.Visible := False;
```

```
TabPanel.Visible := False;
```

بعد ذلك، تحدد عبارة الـ with الـ Alignment الخاص بالـ StatusPanel الرئيسية، وتحدد النص الى خاصية الـ Caption:

```
with StatusPanel do
```

```
begin
```

```
Alignment := taLeftJustify;
```

```
Caption := 'Error #45: Now look what you've done!';
```

```
end;
```

**ملحوظة:** اكتب علامتي تنصيص فرديتين لإدخال رمز علامة التنصيص في string. انظر عبارة تحديد الـ Caption السابقة كمثال.

Note

يمكنك أيضاً تحديد خاصية الـ Alignment في وقت التصميم بالـ Object Inspector. في هذه الحالة، يمكنك ببساطة تحديد الـ Caption string. لاستعادة الـ status panel الى وضعها الطبيعي - بعد إنتهاء الـ Timer، مثلاً، أو يتلقى الـ events الخاص بالفأرة أو لوحة المفاتيح - قم بتعيين الـ null string للـ panel الرئيسية، وحدد خصائص الـ Visible للـ panel الفرعية. على سبيل المثال، قم بتنفيذ هذا الـ code:

```
StatusPanel.Caption := "";
KeyPanel.Visible := True;
TabPanel.Visible := True;
```

### استخدام ال Format function ،

تعد ال Format function الخاصية بـ Delphi مفيدة في إعداد strings للعرض في status panels ، ولكن يمكنك استخدام هذه ال utility function لعرض string-format اخرى . قد تريد مراجعة ال Format online help الخاصة بـ Delphi قبل قراءة الملاحظات والاقتراحات التالية .

إذا كنت على دراية بـ C أو C++ ، فإنك ستعرف على ال Format على انها الموازى لها في ال Pascal ال sprintf() .

قم بتحديد متغيرين أو أكثر للـ Format وقم بتحديد نتيجة ال function الى متغير string ، أو قم بتمريره مباشرة إلى ال non-varstring parameter فى ال procedure أو ال function . يحدد ال Delphi ال Format وال parameters الخاصة بها كما يلي :

```
function Format(const Format: string;
const Args: array of const): string;
```

● **Format**: متغير من نوع string أو literal يحتوى على ال Format specifiers تحتاجه ال Format لإنشاء ال string الناتج .

● **Args**: عبارة عن open array يحتوى على واحداً أو أكثر من arguments - مثل الاعداد الصحيحة ، والمتغيرات floating-point ، characters ، و strings- ليتم إدخالهم فى ال string الناتج . (من الناحية الفنية ، يمكن ان يكون لديك متغيرات أكثر من ال Format specifiers ولكنك تتلقى رسالة خطأ فى وقت التشغيل اذا قمت بتعريف عدد قليل) . قم بوضع المتغيرات فى الاقواس والمتغيرات المتعددة المنفصلة ضعها بين الفاصلات .

**ملحوظة:** تسمح ال Pascal لاسم ال procedure أو ال function بأن يكون هو نفس اسم ال procedure وبذلك يكون اول ال procedure للـ Format مسمى بـ Format . ولكن فى ال function ، فإن تعيين قيمة

Note

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

لإسم الـ function يكون امرأً محيراً إذا كان الـ procedure يستخدم نفس الأسم. في هذه الحالة، يمكنك تعيين نتيجة للكلمة الرئيسية Result.

تعتبر الـ Format مفيدة في إدخال string في strings آخر. على سبيل المثال، قم بتعيين متغير string مثل التالي (إذا اردت الاستمرار، أضف Button و Onclick Label component على الـ form ثم أضف الـ code التالية في الـ Button الخاص بالـ Button، قبل كلمة البداية الرئيسية):

```
var
```

```
Proc: string;
```

قم بتعيين string لـ Proc ثم استخدم الـ Format لإدخال هذا الـ string في آخر (ادخل هذه الـ code بين البداية والنهاية):

```
Proc := 'Pentium II';
```

```
Label1.Caption := Format('I wish my PC had a %s CPU', [Proc]);
```

يحدد السطر الأول literal string لـ Proc. يستخدم السطر الثاني الـ Format لإضافة Proc في literal string، ابدال الـ Format specifier بـ %s بقيمة Proc's. لاحظ ان Proc's محاط بالاقواس لان المتغير الثاني الذي تم تمريره للـ Format هو open array، والذي يشبه مجموعة Pascal من حيث الاستخدام، ولكنه في الواقع قائمة متغيرة من الـ parameters يمكن ان تكون من أى نوع. والنتيجة النهائية هي label:

```
I wish my PC had a Pentium II CPU
```

يمكنك أيضاً استخدام الـ Format لإدخال قيم عديدة في strings. على سبيل المثال، قم بتعريف متغير الـ floating-point:

```
var
```

```
Balance: double;
```

قم بتحديد قيمة للـ Balance، وأدخلها في string باستخدام %8.2f، والذي يحول القيمة الى string في ثمان أعمدة ومكانين عشريين:

```
Balance := 159.72;
```

إن تنفيذ هاتان العبارتان يحدد الـ Caption الخاص بـ Label2 بالـ string التالي (لاحظ أن الأعمدة الثمانية تتضمن النقطة العشرية):

Your balance is \$ 159.72

إن ال Format specifiers يبدأ أولاً بعلامة مئوية يتبعها حرف، ويسمى النوع، وهو يمثل نوع المعلومات التي يتم إدخالها في ال string أيضاً كيفية عرض هذه المعلومات. على سبيل المثال، تشير %d الى قيمة عشرية، و %x الى قيمة سداسية، و %s هو null-terminated string أو Pascal، و %f أو %g يشير الى قيم floating-point. إنها مسئوليتك أن توفر قيمة في ال Args لكل Format specifier. وإنها أيضاً مسئوليتك أن توفر قيم للأنواع المشار إليها.

فيما بين علامة ال % وحرف النوع (d، x، s، أو أى حرف آخر) يمكن ان توجد انواع اخرى من المعلومات، مثل مؤشر الضبط الايسر (-)، قيمة عرض العمود، الدقة (يسبقها نقطة عشرية). على سبيل المثال، يهيئ ال %6.3f قيمة floating-point في ستة اعمدة وثلاث اماكن عشرية. وال %8x ينسق يساراً قيمة عدد صحيح سداسية في ثمان اعمدة.

فكرة: ان Argument index المحاط بالاقواس يبسط عملية تكرار القيم. على سبيل المثال، هذه العبارة:

Format ('%s %d %0:s %d', ['test',10])

تنتج 10 test 10 string test، اذا كان لديك قيم قليلة تحتاج ان يتم وضعها في مناطق متعددة في string، هذا يوفر الذاكرة التي كانت ستستهلك بسبب ال arguments المتكررة. تأكد من ان تحدد عدد ال arguments والتي تكون كافية لكل قيمة محاطة متكررة.

ومزيداً من الامثلة تساعدك ان تستخدم وتفهم ال Format. أولاً، حدد بعض القيم الثابتة المكتوبة (التي تشبه المتغيرات، ولكن لها أنواع بيانات معرفة وقيم افتراضية حسب النظام):

const

XInt: Integer = 123;

XLongInt: Longint = 12345678;

XChar: Char = '@';

XString: string = 'My dog has knees';

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

```
XSingle: Single = 3.14159;  
XDouble: Double = 9.8695877281;  
XExtended: Extended = 9.51413;
```

أضف الـ Label على الـ form لعرض كلاً من هذه القيم . في Button أو event handler آخر ، أدخل العبارات التالية (قد تختلف الـ Names للـ Label الخاص بك عن الخاص بي):

```
Label3.Caption := Format('XInt (dec) = %d', [XInt]);  
Label4.Caption := Format('XInt (hex) = %x', [XInt]);  
Label5.Caption := Format('XLongInt = %d', [XLongInt]);  
Label6.Caption := Format('XChar = %s', [XChar]);  
Label7.Caption := Format('XString = %s', [XString]);  
Label8.Caption := Format('XSingle = %f', [XSingle]);  
Label9.Caption := Format('XDouble = %G', [XDouble]);  
Label10.Caption := Format('XExtended = %G',  
[XExtended]);
```

ان تشغيل البرنامج وضغط الزر يعرض السطور التالية :

```
XInt (dec) = 123  
XInt (hex) = 7B  
XLongInt = 12345678  
XChar = @  
XString = My dog has knees  
XSingle = 3.14  
XDouble = 9.8695877281  
XExtended = 9.51413
```

وايضاً أضف الـ Label لعرض نتيجة الـ string التي تم تهيئتها :

```
Pi=3.141593, Frac=0.141593, Int=3
```

**على القرص المدمج:** يمكن للـ Format ايضاً ان تدخل قيم متعددة في الـ strings . أدخل ما تحتاج من الـ Format specifiers من أى نوع ، وتضع هذه الـ arguments في اقواس وتفصل فيما بينها بالفصلات .



على سبيل المثال ، القائمة (٧-٤) تحتوي على code يمكنك ادخاله فى OnClick الخاص بال Button . النص موجود على القرص المدمج فى دليل ال Source\Format فى ملف Format.pas .

### ايجاد components:

استخدم function وال FindComponent للبحث عن form أو Components array ل object آخر بالاسم . ويرجع هذا ال Components array آخر ، و إنه يقدم طريقة بسيطة لاداء عمليات متعددة على هذه ال objects - إعادة تحديد حجم كل ازرار ال form ، مثلاً على toolbar أو ال status panel . أولاً ، حدد متغيراً من نوع TComponent :

var

C: TComponent;

القائمة (٧-٤): تحديد ال caption الخاص بال Label string تم  
تحديد مواصفاته بقيم arguments متعددة

procedure TForm1.Button1Click(Sender: TObject);

var

R1, R2: Double;

S: string;

begin

R1 := Frac(Pi);

R2 := Int(Pi);

S := Format('Pi= %8.7G, Frac=%8.6F, Int=%G',  
[Pi, R1, R2]);

Label1.Caption := S;

end;

ثم ، لإيجاد component بالاسم ، استخدم عبارة مثل هذه :

C := FindComponent('MyComponent');

إذا لم تكن C لاشئ، فإنها ترجع الى ال component الذى يسمى MyComponent. إذا كانت C لاشئ، فلا يوجد object يحمل الاسم

## الباب السابع: إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

المطلوب. إنك في الغالب تتبع الاستدعاءات للـ FindComponent بعبارة if مثل هذه:

```
if C <> nil then  
    { ... perform action on C here }
```

ومن الأفضل، استخدم (is) في الـ Pascal لتأكيد ان C هو حقاً نوع الـ object الذي تعتقد انه هو الذي يجب ان يكون:

```
if C is TListBox then  
    { ... perform action on ListBox object C here }
```

لاحظ ان العبارة تستخدم اسم component's class والذي يبدأ بـ T. وتقارن عبارة if بين نوع object الـ C واسم component's class. يمكنك استخدام برمجة مشابهة متى تحتاج الى تحديد نوع component's class.

**فكرة:** عند استخدام (is) لاختيار ما اذا كان الـ object الذي رجع بالـ FindComponent هو نوعاً معيناً، فلا يجب عليك ايضاً ان تختبر اذا ما كانت نتيجة الـ function لاشئ. على سبيل المثال، اذا كان التعبير "C is TListBox" حقيقياً، إذن C لا يمكن ان تكون لاشئ.

**على القرص المدمج:** أنظر إلى الـ procedure يستدعى الـ FindComponent لاستخدام الـ code، أضف العديد من الـ check Box على الـ form بالإضافة إلى الـ Button. اضغط مرتين الـ Button واستخدم الـ procedure التالي على أنه الـ OnClick (هذا النص موجود على القرص المدمج في دليل الـ Source\FindComp في ملف (Findcomp.pas). قم بتشغيل البرنامج واضغط بعض الـ CheckBoxes. اضغط الـ Button لجعل كل الـ CheckBoxes on و off.

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
```

```
    C: TComponent;
```

```
    I, J, K: Integer;
```

```
begin
```

```
    K := 0;
```



```

for I := 0 to Form1.ComponentCount - 1 do
begin
  C := FindComponent('CheckBox' + IntToStr(I + 1));
  if C is TCheckBox then
    with C as TCheckBox do
      Checked := not Checked;
    end;
end;

```

يرجع الـ FindComponent، الـ TComponent object، وفي الغالب ايضاً سيكون عليك ان تستخدم بعض البرمجة الإضافية لإخبار الـ Compiler أى نوع من الـ component يكون الـ C فى الحقيقة. يمكنك فعل هذا بعدة طرق. يمكنك استخدام type-cast expression من نوع الـ component وتكون الـ C بين قوسين. على سبيل المثال، هذا يحدد الـ CheckBox بـ True :

```
TCheckBox(C).Checked := True;
```

أو يمكنك استخدام عبارة with لإخبار Delphi ان يتعامل مع الـ C كنوع معين من الـ component :

```

with C as TCheckBox do
begin
  Checked := False;
...
end;

```

ان هذا هو المفضل بشكل عام عن الـ type casting، ان Delphi يقبل تعبير الـ type-cast دون تردد، حتى اذا كان الـ C object ليس من النوع المناسب. ان استخدام الكلمة الأساسية as، اذا كانت الـ C فى هذا المثال ليست object من الـ TCheckBox class، يولد Delphi خطأ الـ Invalid Type Cast.

**تحذير:** كن على حذر عند استخدام الـ FindComponent مع عبارات with فى الـ Pascal. بالرغم من انك ستكون غالباً تريد ان تبحث Component array خاص بالـ form، فإن الـ objects اخرى

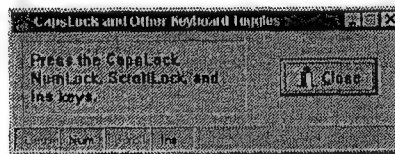


## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

ايضاً لها هذا الـ array وفي داخل عبارة with، قد تنتهي الى بحث قائمة component آخر. لحل هذا النوع من المشاكل، إما أن تستدع الـ FindComponent داخل عبارة with، أو تستخدم تعبيراً مثل MainForm.FindComponent للإشارة الـ object و الـ array Component الذي تريد بحثه.

### : Keyboard status panels

ان واحداً من العناصر الكلاسيكية للـ status panel المنظمة هو مجموعة Pointers لربط لوحة المفاتيح. إنها توضح بشكل نموذجي تحديدات مفاتيح الـ Caps Lock، الـ Num Lock، الـ Scroll Lock، ومفاتيح الـ Ins. ان تطبيق الـ CapsLock، والموضح في شكل (٧-٨)، يظهر هذا التقنية. وتوجد ملفات البرنامج على القرص المدمج في دليل الـ Source\CapsLock. توضح القائمة (٧-٥) الـ source code للـ Main.pas الخاص بالبرنامج. هناك اساليب عديدة ممكنة لإنشاء Pointers لربط لوحة المفاتيح، ولكن هذا الأسلوب الذي اعتقد انه الافضل يظهر نصاً معتمداً للمفاتيح في حالة الإغلاق ونصاً مظلماً (طبيعي) للمفاتيح في حالة الفتح. ولكن، تطبيق هذا مع الـ TPanel objects لا يكون مباشراً بالضبط، كما سأوضح بعد القائمة.



شكل (٧-٨): تطبيق الـ CapsLock يعرض حالات الفتح والإغلاق لمفاتيح الـ CapsLock، الـ Num Lock، الـ Scroll Lock، والـ Ins

القائمة (٧-٥): Capslock\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

---

SysUtils, Windows, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Buttons;

type

```
TMainForm = class(TForm)
    StatusPanel: TPanel;
    CapsLockPanel: TPanel;
    NumLockPanel: TPanel;
    ScrollLockPanel: TPanel;
    InsPanel: TPanel;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    Bevel1: TBevel;
    CapsLockLabel: TLabel;
    NumLockLabel: TLabel;
    ScrollLockLabel: TLabel;
    InsLabel: TLabel;
    procedure FormKeyDown(Sender: TObject; var Key: Word;
        Shift: TShiftState);
    procedure FormActivate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
    procedure UpdateKeyPanel;
end;
```

var

```
MainForm: TMainForm;
```

implementation

```
{ $R *.DFM }
```

```

procedure TMainForm.UpdateKeyPanel;
begin
    CapsLockLabel.Enabled :=
        GetKeyState(VK_CAPITAL) and 1 = 1;
    NumLockLabel.Enabled :=
        GetKeyState(VK_NUMLOCK) and 1 = 1;
    ScrollLockLabel.Enabled :=
        GetKeyState(VK_SCROLL) and 1 = 1;
    InsLabel.Enabled :=
        GetKeyState(VK_INSERT) and 1 = 1;
end;

procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;
    Shift: TShiftState);
begin
    UpdateKeyPanel;
end;

procedure TMainForm.FormActivate(Sender: TObject);
begin
    UpdateKeyPanel;
end;

end.

```

لإنشاء status panel مرتبطة بلوحة المفاتيح ، اتبع الخطوات التالية :

- ١ - أضف ال TPanel object على ال form وقم بتعديله كما هو موضح في هذا الباب ، على سبيل المثال ، حدد ال Height بـ ٢٥ واختر اسلوب ال 3D .
- ٢ - قم بمحاذاة ال panel بأسفل النافذة .
- ٣ - أضف ال subpanels في ال TPanel الرئيسية ، واجعل اسماءها CapsLockPanel ، NumLockPanel ، الى آخره .

#### ٤- احذف كل ال Captions الخاصة بال panel .

ان عرض dim text فى ال TPanel object [انظر شكل (٧-٨)] يعد اقل من غير المعقد لان هذا ال component لا يعتم نص ال Caption الخاص به عندما تكون خاصية ال Enabled لل TPanel محددة بـ False . وبسبب هذا الحد، يدخل تطبيق ال CapsLock اربعة Label objects فى كل subpanel . لعرض ال label لمفتاح يكون فى حالة إغلاق، يحدد البرنامج خاصية ال Enabled لل Label بـ False ؛ لعرض ال normal label لمفتاح فى حالة فتح، يحدد البرنامج ال Enabled بـ True . لا يقوم البرنامج بتعيين نص لل Captions الخاصة بال subpanel .

يوضح ال UpdateKeyPanel Procedure كيفية الحصول على المواصفات الحالية لمفاتيح ال Caps Lock ، Num Lock ، Scroll Lock ، و Ins . يستدعى ال procedure و Windows API function وهى ال GetKeyState ، والى يمكنك ان تمرر إليها أى virtual key code ظاهرى ونتيجة تنفيذ ال GetKeyState عدد صحيح . اذا كان اقل بت فى هذه القيمة يساوى 1 ، يكون المفتاح مفتوحاً؛ وان لم يكن كذلك، يكون المفتاح مغلقاً . ويختبر تعبير AND المنطقى هذه البت، معطياً النتيجة فى قيمة Boolean True أو False . ان تعيين هذه القيمة لخاصية ال Enabled لل Label يعرضها بنمط عادى أو معتم .

لعرض قيم المفاتيح بشكل سليم عندما يبدأ البرنامج وعندما يضغط المستخدمون واحداً من ال labeled keys ، يقوم ال CapsLock بإنشاء إثنين من ال event handler لل form وهما OnActivate و OnKeyDown . وكلا من هذين البرنامجين يستدعى ال UpdateKeyPanel procedure لتحديث ال status ال panel .

**فكرة:** لفحص تغيرات مفاتيح ال Caps Lock ، Num Lock ، Scroll Lock ، و Ins فى OnKeyDown الخاص بال form ، حدد خاصية ال KeyPreview لل form بـ True .

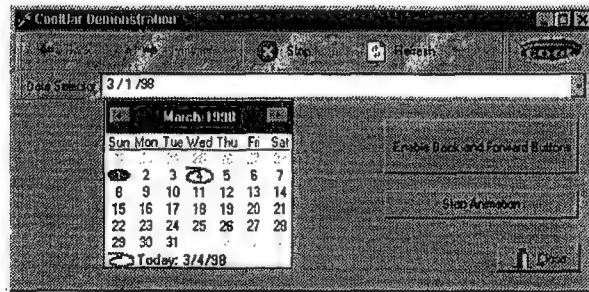
Tip



## الـ Coolbars :

الـ Coolbars، والذي يعرف أيضاً بالـ rebar، هو objects . بإمكانه إحتواء أكثر من objects أخرى . وتلك الـ objects، والتي يمكن أن تكون objects من classes مشتقة من الـ TWinControl، تستقر على واحدة أو أكثر من الـ bands والتي تقسم الـ Coolbar الى قطاعات . وكل bands في حد ذاته يعد object من الـ TCoolBand class . في وقت التشغيل، يمكن للمستخدمين ضغط وسحب الـ bands لخلطها وتجميعها لجعل عرض الـ Coolbar يتفق مع رغبتهم . اثناء تصميم البرنامج، يمكنك فعل نفس الشيء لترتيب الـ Coolbar bands وفق رغبتك .

**على القرص المدمج:** يوضح شكل (٧-٩) مثال Coolbar من برنامج العرض الخاص بهذا الفصل، وهو CoolDemo، الموجودة في الموجود في دليل Source\Cooldemo على القرص المدمج . ومثال الـ Coolbar له two bands، واحداً فوق الآخر . ويحمل الـ band العلوي اربعة ازرار، والذين لا يؤديون أيه اعمال حقيقة في العرض - إنهم للعرض فقط . ويحمل الـ band الثاني object DateTimePicker، ويعرض أيضاً "Date selector." label.



شكل (٧-٩): يوضح تطبيق الـ CoolDemo كيفية إنشاء واستخدام الـ Coolbar، والذي في هذه الحالة، يحمل اربعة ازرار و DateTimePicker object

سوف تصبح الـ Coolbars أكثر إنتشاراً عندما يحصل الـ Windows 98 على القبول اللازم . اذا كنت تراجع تطبيق Delphi موجود بالفعل، فقد تريد

التفكير في ابدال ال toolbars الخاصة بك بـ Coolbars. وهذا يعطى برامجك مظهراً ثابتاً بين برمجيات ال Windows الأخرى. يوضح هذا الفصل، كيفية تقسيمه الى bands، كيفية إضافة انواع متنوعة من components الى تلك ال bands.

**ملحوظة:** للحصول على مثال آخر عن ال Coolbar، انظر برنامج متصفح Web في دليل Demo\Coolstuff الخاص بـ Delphi.



### إنشاء ال Coolbars :

إنشاء ال Coolbar هو أمر غاية في السهولة- فقط أضف ال component ال Coolbar من Win32 VCL palette على ال from حسب النظام الافتراضى، يلبق ال Coolbar نفسه على قمة النافذة. يمكنك تغيير هذه القيمة المتغيرة باختيار قيمة مختلفة لخاصية ال Align لل Coolbar، ولكن فى معظم الحالات، يكون قمة النافذة، تحت menu bar مباشرة ان وجد، وهى المكان الذى يجب عرض ال Coolbar فيه، كما هو موضح فى شكل (٧-٩).

**على القرص المدمج:** توضح القائمة (٧-٦) source code لل CoolDemo فى ملف Main.pas، الموجود فى دليل Source\CoolDemo على القرص المدمج. سوف اشرح البرمجة الموجودة فى هذه القائمة فى الاماكن المناسبة فى هذا الفصل والفصول التالية.



#### القائمة (٧-٦): CoolDemo\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics,  
Controls,
```

```
Forms, Dialogs, ComCtrls, ToolWin, Images, StdCtrls, Buttons;
```



type

```
TMainForm = class(TForm)
  Coolbar1: TCoolbar;
  ToolBar1: TToolBar;
  NavigatorImages: TImageList;
  NavigatorHotImages: TImageList;
  DateTimePicker1: TDateTimePicker;
  ToolButton1: TToolButton;
  ToolButton2: TToolButton;
  ToolButton3: TToolButton;
  ToolButton4: TToolButton;
  ToolButton5: TToolButton;
  BitBtn1: TBitBtn;
  BitBtn2: TBitBtn;
  Animate1: TAnimate;
  Button1: TButton;
  procedure BitBtn1Click(Sender: TObject);
  procedure Button1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

var

```
MainForm: TMainForm;
```

implementation

```
{ $R *.DFM }
```

```
procedure TMainForm.BitBtn1Click(Sender: TObject);
```

var



```

TF: Boolean; // True or False flag
S: String;
begin
    TF := ToolButton1.Enabled;
    ToolButton1.Enabled := not TF;
    ToolButton2.Enabled := not TF;
    if TF
    then S := 'Enable'
    else S := 'Disable';
    BitBtn1.Caption := S + ' Back and Forward Buttons';
end;

procedure TMainForm.Button1Click(Sender: TObject);
begin
    Animate1.Active := not Animate1.Active;
    if Animate1.Active
    then Button1.Caption := 'Stop Animation'
    else Button1.Caption := 'Begin Animation';
end;

end.

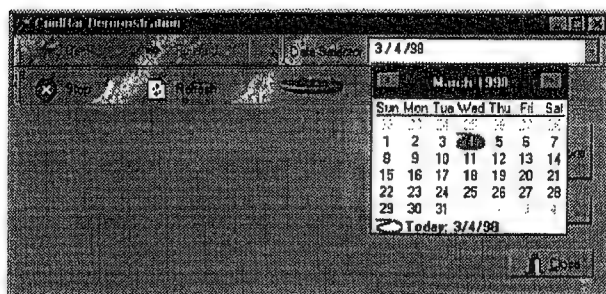
```

حتى تعتاد على الـ Coolbar ، قم بتشغيل الـ CoolDemo الآن بتحميل ملف مشروع الـ Cooldemo.dpr في Delphi من دليل Source\Cooldemos ، واضغط F9 . اضغط الزر الكبير لتشغيل وإبطال الزرين الأولين في الـ bands العلوى للـ Coolbar . لاحظ انه عندما تحرك مؤشر الفأرة فوق الزر ، فيبدو وكأنه يكبر عن الشاشة وتتغير ايقونته من الاسود والايض الى صورة ملونة . إنك توفر هذه الصور باستخدام واحداً أو أكثر من ImageList objects ، كما اوضحت قبل ذلك في هذا الفصل .

لإعادة تشكيل تنظيم الـ Coolbar ، اضغط واسحب واحداً من اثنان من الـ bands الرأسين (إنها يبدو ان لى كالمسمارين المستديرين) فى اقصى اليسار فى الـ Coolbar band . على سبيل المثال ، التقط الـ band السفلى وحركه تجاه القمة-

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

هذا يجمع الـ bands ويجعلها يجلسان جنباً الى جنب كما هو موضح فى شكل (٧-١٠). " قد يكون عليك ان تجرب عدة مرات لتحصل على التأثير الذى تريده ).



شكل (٧-١٠)؛ اضغط واسحب bands الرأسى لـ Coolbar band لإعادة ترتيب

تنظيم الـ Coolbar كما هو موضح هنا باستخدام تطبيق الـ CoolDemo

لكل band فى الـ Coolbar لديها text label . لإنشاء label ، اضغط الزر البيضواوى الواقع بعد خاصية الـ Bands Coolbar object . وهذا يفتح الـ Coolbar Bands Editor . اختر أى Band (أو أضف واحداً اذا لزم الأمر بضغط زر الـ الخاص Add New) ، ثم ادخل الـ label الخاص بك فى خاصية الـ Text باستخدام الـ Object Inspector التابع بـ Delphi .

**ملحوظة:** الطريقة الوحيدة للتوصل الى band objects فردية هى من خلال الـ Coolbar Bands Editor . بالرغم من ان كل band object من الـ TCoolBand class ، الا ان objects غير مقرر بصورة فردية فى الـ form class ولذلك ، فإنها غير واردة فى قائمة لائحة الـ Object Inspector .



### خصائص الـ Coolbar:

فيما يلى بعض الملاحظات حول خصائص الـ Coolbar . المختارة . قد تحتاج الى قراءة هذا الفصل والفصول القليلة التالية قبل ان تفهم هدف بعض هذه الخصائص :

• **Align:** فى المعتاد، تستقل الـ Coolbar على قمة النافذة، ولكن يمكنك تحديد الـ Align بـ alBottom ، alRight ، أو alLeft لتغيير التحديد الافتراضى بـ alTop . وقد تحدد بعض التطبيقات هذه الخاصية بـ alNone للحصول على

Coolbar ذى حجم ثابت، أو alClient للملء النافذة بخلفية ال Coolbar. عندما تكون ال Align بـ alTop أو alBottom، تكون ال Vertical محددة بـ False. عندما تكون ال Align بـ alLeft أو alRight، تكون ال Vertical محددة بـ True. كن على حذر من هذا التفاعل فيما بين الحقلين.

• **AutoSize**: انك دائماً تحدد هذه الخاصية بـ True، وهى قيمتها الافتراضية، حيث يقوم ال Coolbar تلقائياً بتعديل حجمه ليتلائم داخل حدود النافذة. ولكن يمكنك ان تحدد هذه الخاصية بـ False اذا كنت تريد ان تضع ال Coolbar يدوياً.

• **Bands**: اضغط مرتين هذه القيمة ال Coolbar Bands Editor. إستخدم ال Editor لإضافة، حذف، إعادة ترتيب ال Coolbar Band، الذى يعتبر كلاً منهما object من ال TCoolBand class. يعتبر ال Bands Editor هو الطريق الوحيد الذى يمكنك بواسطته التوصل الى Bands منفردة فى وقت التصميم. اختر Band باستخدام ال Editor ثم ادخل قيم الخاصية فى نافذة ال Object Inspector كما تفعل مع ال components الأخرى.

• **Bitmap**: قم بتعيين أى ملف bitmap لهذه الخاصية لتلوين خلفية ال Coolbar. من الناحية النموذجية، يجب ان تكون ال bitmap متماثلة، بحيث عندما تتكرر فوق ال Coolbar، يكون التأثير البصرى الأخير بلا جوانب. من الأفضل ان تستخدم صور ذات ألوان فاتحة أو رمادية فاتحة - لان الألوان الداكنة تجعل من الصعب رؤية النص والازرار الموجودة على ال Coolbar Bands. اذا لم تكن تريد استخدام bitmap image لخلفية ال Coolbar، يمكنك تحديد خاصية ال Color لتنظيم العرض وإذا لم تحدد Bitmap أو قيمة Color، فإن ال Coolbar له نفس لون واجهة الزر العادى ال (Color = clBtnFace).


• **EdgeBorders**: وهذه الخاصية لها اربع قيم فرعية التى يمكنك تحديدها بالضغط مرتين على علامة الزائد الصغيرة الى اليسار من ال EdgeBorders. فى نافذة ال Object Inspector. ولكل قيمة تحدد بـ True أو False - قم بتجربة مجموعات متنوعة لترى تأثيراتها. والقيم الاربعة الفرعية هى: ebLeft، ebRight، ebTop، ebBottom لإنشاء Coolbar ذى حدود غير مرئية، حدد القيم الفرعية الأربع بـ False، أمحو خاصية ال Bitmap، وحدد ال Color بـ

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

clBtnFace سيكون للـ Coolbar الناتج غير مرئياً في وقت التشغيل، ولكنه يعمل وكأنه واحداً ذا حدوداً مرئية.

● **ShowText**: لعرض الـ Labels في الـ Coolbar's bands، اختر الـ bands باستخدام الـ Bands Editor وادخل الـ label الخاص بك في خاصية الـ Text باستخدام الـ Object Inspector. حدد خاصية الـ ShowText للـ Coolbar بـ True لعرض الـ label band. وتعرض هذه أيضاً في الـ Bands Editor.

● **Vertical**: تؤثر هذه الخاصية على كيفية عرض الـ bands في الـ Coolbar. حدد الـ Vertical بـ False لعرض الـ bands أفقياً، واحداً فوق الآخر (وهو التحديد الافتراضي). حدد الـ Vertical بـ True لعرض الـ bands رأسياً، جنباً إلى جنب. حدد خاصية الـ Align للـ Coolbar، والتي تؤثر على موقع النافذة، قبل تغيير الـ Vertical. يعتبر هذا ضرورياً لأن تعيين قيم خاصية الـ Align قد يغير قيمة الـ Vertical. (راجع الـ Align).

**Tip**  **فكرة**: لأن الـ Coolbar غالباً ما يحمل الـ bands لـ controls أخرى عديدة، فإنه من الصعب عادة أن تختار الـ Coolbar object بالفأرة في Delphi. إذا حدث هذا، استخدم قائمة لائحة الـ Object Inspector لاختيار الـ object - مثلاً، افتح تطبيق الـ CoolDemo واختر الـ Coolbar1 لفحص قيم خاصية الـ Coolbar الخاص بالعرض.

### خصائص الـ CoolBar:

فيما يلي ملاحظات حول خصائص الـ CoolBand، وتذكر أن الطريقة الوحيدة للوصول إلى الـ band objects منفردة هي ضغط الزر البيضوي الواقع بعد خاصية الـ Bands للـ Coolbar، وان تفتح الـ Bands Editor. يمكنك عندئذ إبراز الـ Band منفرد واستخدام الـ Object Inspector لتحديد خصائصه. ولأن الـ Coolbar يملك الـ CoolBand objects، فإنها لا تظهر في قائمة لائحة الـ components الخاصة بالـ Object Inspector.

● **Bitmap**: كل الـ Band من الممكن أن يملك الـ Bitmap image خاصه به للعرض في خلفيتها. إذا قمت بتعيين ملف الـ Bitmap لهذه الخاصية، فإنها تغطي

على أى قيمة Bitmap معينة للـ Coolbar . بدلاً من استخدام الـ Bitmap ، يمكنك أيضاً تغيير خاصية الـ Color لأى Band فردى .

• **Break** : حدد هذه الخاصية بـ True لوضع الـ band على السطر جديد على الجانب الأيسر من الـ Coolbar عندما تحدد الـ Break بـ False ، يتم عرض الـ band بعد أى band سابق ولا يبدأ سطر جديد .

• **Control** : يمكن أن يحمل كل band ، Control واحد . فى العادة ، يتم تحديد هذه الخاصية تلقائياً للـ component الذى تسقطه على الـ Coolbar ، ولكن يمكنك تحديد الـ Control لأى classobject مشتقة من الـ TWinControl .

• **FixedBackground** : فى المعتاد ، حدد هذه الخاصية بـ True بحيث تكون الـ Bitmap التابعة للـ Coolbar معروضة باستمرار خلف كل الـ bands . عندما تكون الـ FixedBackground محددة بـ False ، تكون الـ Bitmap التابعة للـ Coolbar معروضة بشكل منفصل لكل bands واعتماداً على نوع الـ Bitmap التى تستخدمها ، قد يكون هذا نافعاً . ولكن للحصول على مظهر بلا جوانب ، يجب أن تكون الـ FixedBackground محددة بـ True . انظر أيضاً خاصية الـ ParentBitmap .

• **FixedSize** : عندما تكون هذه الخاصية محددة بـ True لا يستطيع المستخدمون إعادة تحديد حجم الـ bands . فى العادى ، تكون هذه الخاصية بـ False (وهو التحديد الافتراضى) .

• **HorizontalOnly** : فى الحالات التى يجب فيها عرض band معين أفقياً فقط ToolBar ، على سبيل المثال - حدد هذه الخاصية بـ True . اذا كانت خاصية الـ Vertical للـ Coolbar الأكبر محددة أيضاً بـ True ، سيصبح الـ bands غير مرئياً . قد تستخدم هذه الخاصية لإخفاء bands مختارة عندما تكون جميع الـ bands مرتبة رأسياً ، ولكن من الصعب تخيل الاستخدام العملى لهذا التحديد الغامض . لغالبية التطبيقات ، استخدم التحديد الافتراضى بـ False .

• **Image Index** : قد يعرض الـ band صورة ايقونة بتحديد هذه الخاصية بقيمة فهرس الصورة فى الـ ImageList object الخاص بها . يجب أن يتم تعيين من object لخاصية الـ Images للـ Coolbar .

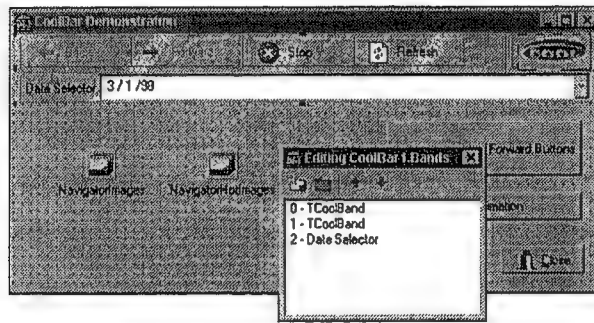
## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

● **ParentBitmap** : عندما تحدد هذه الخاصية بـ True، ويعرض الـ bands نفس الـ Bitmap image المعينة للـ Coolbar . bands parent . ان تعيين صورة لخاصية الـ band's Bitmap يحدد تلقائياً الـ ParentBitmap بـ False .

### إضافة ToolBars للـ Coolbars :

بعد اسقاط الـ Coolbars على الـ form سوف تريد ان تقسمه الى bands .  
يمكنك ان تفعل هذا بطريقتين :

● اضغط الزر البيضاوى الواقع بعد خاصية الـ Bands للـ Coolbar فى نافذة الـ Object Inspector . هذا يؤدى الى فتح الـ Coolbar Bands Editor [انظر شكل (٧-١١)]، والذي يمكنك استخدامه لإضافة، وحذف، وإعادة تنظيم الـ Bands.



شكل (٧-١١): استخدام الـ Coolbar Bands Editor لإنشاء، حذف وإعادة ترتيب الـ band فى الـ Coolbar

● قم باسقاط الـ windowed control مثل الـ ComboBox، الـ ToolBar، أو مثل ما هو فى برنامج العرض، الـ DateTimePicker فى الـ Coolbar . وهذا ينشئ تلقائياً band جديد لحمل الـ control .

ان غالبية الـ Coolbar تعرض زراً واحداً أو أكثر لإنشاء الـ ToolBar . لفعل هذا، قم أولاً باسقاط الـ Coolbars على الـ form، ثم اسقط الـ ToolBar object (موجود أيضاً على Win32 palette) على الـ Coolbar . اضغط الـ ToolBar باستخدام زر الفأرة الأيمن، واختر أمر الـ New Button لإنشاء ازرار الـ ToolBar . يعتبر كل زر object من الـ TToolBar class (وهذه ليست على الـ VCL

palette- يجب ان تستخدم امر ال New Button لإنشاء هذه الانواع من الازرار).

**Tip فكرة:** للحصول على مظهر Toolbar كلاسيكى، اتبع هذه الافكار:  
حدد خاصية ال AutoSize لل Toolbar بـ True، وحدد خاصية ال  
AutoSize لكل ال ToolButton بـ False. حدد خاصية ال Flat لل  
Toolbar بـ True. لتعديل احجام الزر، اختر ال Toolbar وعين قيم لخصائص  
ال ButtonHeight وال ButtonWidth. يجب ان تكون جميع الازرار فى ال  
Toolbar ذات حجم واحد، والذي يتم تحديده بواسطة ال Toolbar، وليس  
بالازرار المنفردة، كما هو الحال فى اغلب ال Delphi components الأخرى.

تبدو ال Coolbars أكثر تميزاً بالخلفية التى تشبه صورة حجرية فى ال  
CoolDemo [راجع شكل (٧-٩)]. وهذا هو الافضل مع ال Toolbar التى تم  
تحديد خاصية ال Flat لها بـ True، الأمر الذى يجعل الخلفية لامعة خلال ازرار ال  
Toolbar. لإنشاء التأثير البصرى فى ال CoolDemo، قمت بتعيين خاصية  
ال Bitmap لل Coolbar للصورة الموجودة على القرص المدمج فى صورة  
Source\Data\Background.bmp. يمكنك استخدام أى Bitmap  
بال Coolbar، ولكن المثال مصمم خصيصاً لتكرار أى خلفية بلا جوانب من أى  
حجم.

والمرة الاولى التى تضيف فيها ازرار لل Toolbar، سوف تكون خاوية.  
يوضح فصل "الصور والصور الهامة" فى هذا الباب كيفية إضافة ايقونة لازرار ال  
Toolbar. لعرض text label- والذي يمكنك فعله ايضاً بعرض ايقونات أو بدون  
عرضها- اختر ال ToolButton، ادخل ال object الخاص بك فى خاصية ال  
Caption باستخدام ال Object Inspector. لكى تجعل ال Caption مرئياً، اختر  
ال Toolbar وحدد خاصية ال ShowCaptions بـ True.

**ملحوظة:** يمكنك إضافة ال Toolbar مباشرة على ال form- لا يجب ان  
يستقر ال Toolbar على ال Coolbar band. ان تقنيات البرمجة هى نفس  
تقنيات ال Toolbar المستقلة كما هو موضح هنا لتطبيق ال

CoolDemo.

### خصائص الـ Toolbar:

فيما يلي ملاحظات حول الخصائص المختارة للـ Toolbar. تذكر، بالرغم من ان هذا الفصل يشرح الـ ToolBars بارتبطها بالـ Coolbars، يمكنك إضافة الـ Toolbar مستقلة على الـ form. لا يجب عليها ان تبقى داخل الـ Coolbars.

● **Align**: ان اغلب الـ Toolbars تعرض على قيمة النافذة، ولكن يمكنك تغيير هذه الخاصية لمحاذاة القضيب على الحد السفلى للنافذة (alBottom)، أو الحافة اليسرى (alLeft)، أو (alRight). للحصول على toolbar أو floating أو ذو حجم ثابت، حدد الـ Align بـ alNone. حددها بـ alClient للملء الـ client area بالنافذة - وهي تقنية جيدة لواجهات التطبيق التي تزدحم بمجموعات الازرار، مثل نظام الادخال المنظم لشاشة اللمس.

● **AutoSize**: عندما تحدد هذه الخاصية بـ True، يعدل الـ Toolbar حجمه تلقائياً ليدخل ازراره دون مسافة مهدرة. إنك في الغالب تريد تغيير هذه القيمة الى True (فهى False حسب النظام الافتراضى) فى غالبية التطبيقات.

● **FuttonHeight, ButtonWidth**: عين قيم لهاتين الخاصيتين قبل إنشاء ازرار الـ ToolBars. جميع الازرار فى الـ ToolBars لها نفس الحجم. يتم التعبير عن القيم بالـ pixels.

● **DisabledImages**: قم بتعيين الـ ImageList لهذه الخاصية، وعين قيم الـ ImageIndex لكل من ازرار الـ ToolBars. تعرض الازرار التى تم ابطالها الايقونات من الـ ImageList object هذا. اذا لم تعين الـ ImageList لهذه الخاصية، يتم إعتام ايقونات الزر المبطل تلقائياً باستخدام الـ Bitmap من خاصية الـ Images (اذا كانت معينة).

● **Flat**: للحصول على مظهر كلاسيكى يشبه متصفح Web، وبخاصة عند إدخال الـ Toolbar فى الـ Coolbar، فإنك تريد ان تحدد هذه الخاصية بـ True. قم ايضاً بتعيين الـ Bitmap للـ Coolbar، وحدد خاصية الـ Transparent للـ Toolbar بـ True. يأخذ الـ Toolbar عندئذ مظهر الـ Coolbar، كما تفعل ازار الـ Toolbar.



● **HotImages**: قم بتعيين ImageList لهذه الخاصية، وعين قيم ImageIndex لكل من ازرار ال Toolbar. يتم عرض هذه الايقونات عندما يقوم المستخدم بتحريك مؤشر الفأرة فوق الازرار. وبالطبع، ان الايقونات هي نفسها التى فى خاصية ال Images، ولكنها مختلفة عندما تشير إليها مؤشر الفأرة. ان الصور الهامة لا تستخدم للازرار التى تم ابطالها (تلك التى لها خاصية ال Enabled محدد ب False).

● **Images**: قم بتعيين ImageList لهذه الخاصية، وعين قيم ImageIndex لكل من ازرار ال Toolbar. يتم عرض هذه الايقونات على الازرار الا اذا كانت مبטلة أو كان المستخدم يشير الى زر باستخدام الفأرة (راجع خصائص ال DisabledImages وال HotImages).

● **Indent**: زد هذه القيمة لعرض مزيد من المسافة عند الحافة اليسرى لل Toolbar قبل الزر الأول.

● **ShowCaptions**: حدد هذه القيمة ب True لعرض خصائص ال Caption للزرار فى ال Toolbar.

● **Transparent**: حدد هذه الخاصية ب True لظهار لون أو bitmap لل object الرئيسى كخلفية لل Toolbar. عند إدخال Toolbar فى Coolbar، فإنك غالباً سوف تحددها وخاصية ال Flat ب True للحصول على المظهر.

● **Wrappable**: عندما تحدد هذه الخاصية ب True، تنتقل ازار ال Toolbar إلى صفوف جديدة اذا لم تكن جميعها ملائمة داخل حدود ال Toolbar. ان ال button objects الفردية يجب ان يكون لديها خصائص Wrap ايضاً محددة ب True حتى يعمل هذا التأثير بشكل سليم.

### خصائص ال ToolButton:

فيما يلى ملاحظات حول الخصائص المختارة لل ToolButton. لإضافة زر الى ال Toolbar، اضغط زر الفأرة الأيمن واختر أمر ال New Button من القائمة. لاختيار زر منفرد، اضغط صورته فى ال Toolbar، أو يمكنك اختيار ال ToolButton بالاسم من نافذة ال Object Inspector.

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

● **AllowAllUp**: حدد هذه بـ True للسماح لكل أزرار مجموعة ما أن تكون غير مختارة. عندما تمحدد بـ False، يجب أن يكون أحد الأزرار المجموعة مختاراً (ويجب أن يتم اختياره أولاً بتحديد خاصية الـ Down له بـ True). وهذه الخاصية ليس لها تأثير إلا إذا كان الـ Style الخاص بالزر محدد بـ tbsCheck، وخاصية الـ Grouped له محددة بـ True.

● **AutoSize**: عندما تمحدد هذه الخاصية بـ True، يقوم الزر تلقائياً بتعديل حجمه ليتلائم في الـ ToolBar. لغالبية التطبيقات، رغم ذلك، يمكنك أن تترك هذه القيمة محددة بـ False وتمحدد خاصية الـ AutoSize للـ ToolBar بـ True. وهذان التحديدان يعطيانك في الغالب التحكم الذي تريده على أحجام الأزرار.

● **Caption**: قم بتعيين label قصيرة لهذه الخاصية، وحدد حقول الـ ShowCaptions للـ ToolBar بـ True، لعرض تعليقات في الأزار.

● **على القرص المدمج: Command**: قم بتعيين أمر من الـ CommandList للمشاركة مع الزر في الـ event handler الخاص بهذا الأمر. وهذا هو الـ event handler الذي ينفذ عندما يضغط المستخدم الزر. أو، يمكنك تعيين event handler للـ OnClick الخاص بالـ ToolButton (كما تفعل بالضبط مع الـ Button objects العادية). ولكن باستخدام الـ CommandList تبسط عملية تشارك الأوامر فيما بين objects متعددة - popup menu، قائمة رئيسية، وزر ToolBar، مثلاً. أنظر برنامج العرض Commands على القرص المدمج المرفق بهذا الكتاب في دليل الـ Source\Commands، وكذلك شرح الـ CommandList component في الباب الخامس.



● **Down**: تكون هذه الخاصية True إذا كان الزر في حالته السفلية (مضغوطاً). عند تجميع أزرار متعددة لها Styles محددة بـ tbsCheck وخصائص AllowAllUp محددة بـ True، فعلى الأقل يجب أن يكون زرراً واحداً في المجموعة له خاصية Down محددة بـ True. يمكنك أن تفعل هذا باستخدام الـ Object Inspector، أو بتعيين True لخاصية الـ Down للزر من خلال جملة تكتب في برنامج. والمكان المناسب لعمل هذا هو OnCreate event handler للـ form.

● **Enabled:** حدد هذه الخاصية بـ True لعرض الزر في مظهره العادى، وكذلك لتشغيل الـ OnClick events له. حددها بـ False لعرض الزر مستخدماً ايقونته اذا كان قد تم تعيين واحدة له. (انظر خصائص الـ DisabledImages والـ Images للـ ToolBar).

● **Grouped:** عين True لهذه الخاصية لإنشاء مجموعات من الازرار تعمل بالتفاعل مع بعضها البعض. يجب ان يكون للازرار Styles محددة بـ tbsCheck. حدد الـ AllowAllUp بـ True اذا كانت جميع الازرار مسموح له بأن تبقى غير مختارة؛ والا، فعلى الأقل يجب ان يكون زرّاً واحداً من المجموعة مختاراً فى كل الاوقات (أى، يجب ان تكون خاصية الـ Down له محددة بـ True).

● **Image Index:** تشير هذه القيمة الى أى من الايقونات يجب عرضها من الـ ImageList المعين لخاصية الـ Images للـ ToolBar (وكذلك خاصيتين الـ DisabledImages والـ HotImages له اذا كانتا مستخدمين). يكون فهرس الصورة الأولى دائماً صفر، والثانية ١، والثالثة ٢ والى آخره. حدد هذه الخاصية بـ 1- اذا لم تكن تريد عرض صورة ايقونة على الزر.

● **Indeterminate:** عندما تحدد هذه الخاصية بـ True، يتم عرض الزر باستخدام لوناً رمادياً، ولكن اذا كانت الـ Enabled ايضاً محددة بـ True، يمكن ان يبقى الزر مختاراً. ان هدف هذا الحقل هو توفير دليل بصرى للمستخدمين ان الزر يمكن ان يكون مختاراً، ولكن قد يكون من غير الصواب ان تفعل هذا بسبب الحالة الراهنة للبرنامج - التحول الى عملية اخرى، مثلاً، اثناء عملية طباعة طويلة. يمكنك ايضاً فحص هذا المجال فى . OnClick event handler للزر، واذا كانت الخاصية True، قم بعرض رسالة تأكيد أو تحذير ان المستخدم قد إختار أمراً غير مناسب.

● **Style:** استخدام هذه الخاصية للاختيار من بين ازرار مختلفة الانواع. ويساوى الـ Style الخاص بالزر العادى tbsButton، وهو التحديد الافتراضى حسب النظام. حدد كل الـ Styles الخاصة بالازرار المجموعة بـ tbsChecked- هذا لا يعرض علامة صح، ولكنه ينشئ زر لاصق يبقى الى اسفل عندما يتم ضغطه

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

حتى يضغط مرة ثانية . يمكنك ايضاً استخدام هذا الاسلوب لإنشاء ازرار خيار مستقلة- لا يجب ان يتم تجميعها (انظر ايضاً الفقرة التالية) .

• **Wrap**، تحدد هذه الخاصية بـ True لتمكن هذا الزر من الانتقال إلى على صف جديد اذا لم تكن لتناسب حدود الـ ToolBar . يجب ايضاً ان تكون خاصية الـ Wrappable للـ ToolBar محددة بـ True حتى يعمل هذا التأثير بشكل سليم .

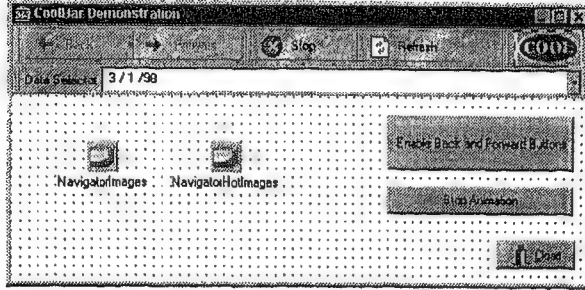
**Tip** **فكرة:** لإنشاء فاصل بصري بين الازرار الأخرى ، حدد الـ Style بـ **tbsDivider** أو **tbsSeparator** (كلاهما نفس الشيء) . هذا النوع من الزر هو للعرض الخالص ولا يمكن ضغطه . لعرض سهم مشير الى اسفل على اليمين داخل الزر ، حدد كل الـ Styles بـ **tbsDropDown** . هذا لا يؤدي إلى إنشاء قائمة للزر- لفعل هذا ، أضف **PopupMenu** من **Standard palette** على الـ form لضغط الـ object مرتين لإنشاء اوامر قائمة ، وعين **PopupMenu** object لخاصية الـ **DropDownMenu** للزر . ان ضغط هذا النوع من الزر في وقت التشغيل يؤدي إلى فتح القائمة التي يمكن للمستخدمين ان يختاروا منها الأوامر .

### الصور والصور الهامة،

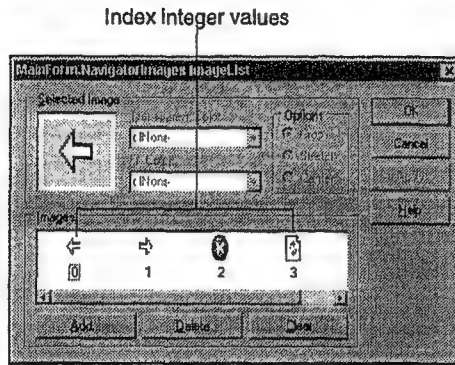
إن الأزرار الموجودة في الـ ToolBar الخاص بالـ Coolbar يجب أن تعرض جرافيك أيقوني يشير إلى ما يفعله الزر . وهذا يتطلب استخدام component آخر من الـ Win32 palette ، ألا وهو الـ **ImageList** . أدخل واحداً أو أكثر من الـ objects على الـ form ، ثم استخدم الـ objects للتحكم في واحدة أو أكثر من الصور الأيقونية للعرض على أزرار الـ ToolBar . (يمكنك استخدام الـ **ImageList** لأغراض أخرى كذلك عندما تحتاج وعاء تخزين لحمل الأيقونات) .

يوضح شكل (٧-١٢) إثنين من الـ **ImageList** في نافذة الـ CoolDemo داخل Delphi . و **ImageList** - ويسميان **NavigatorImages** و **NavigatorHotImages** - يظهران في وقت التصميم كأيقونات ، ولكنهما غير مرئيان عندما يبدأ البرنامج في التشغيل . لكي أجعل الأيقونات أكثر رؤية هنا ، فقد حددت خلفية الـ form باللون الأبيض ؛ على شاشتك ستكون الخلفية ملونة بشكل

طبيعى إعتماًداً على تحديدات عرض ال Windows mpct الخاصة بك . لإضافة ، وحذف ، وإعادة ترتيب أيقونات ال ImageList ، اضغط مرتين ImageList object . هذا يفتح نافذة محرر ال ImageList الموضحة فى شكل (٧-١٣) .



شكل (٧-١٢): ال form فى وقت التصميم لتطبيق ال CoolDemo توضح إثنين من ال ImageList objects كإيقونات والتي تصبح غير مرئية عندما يبدأ البرنامج فى التشغيل



شكل (٧-١٣): افتح نافذة محرر ال ImageList بالضغط مرتين على ال object ImageList component

استخدم زر ال Add فى محرر ال ImageList لفتح أى أيقونة أو ملف bitmap آخر ، وأدخل الصورة فى القائمة . لتغيير ترتيب الصور الموجودة فى القائمة ، اضغطها واسحبها للمواضع التى تريدها . اختر أى صورة واضغط Delete لحذفها من القائمة . اضغط زر ال Clear لمسح كل الصور .

**ملحوظة:** يستطيع محرر ال ImageList أن يعدل ألوان ال Transparent وال Fill color لل bitmaps أيضاً أن crop أو stretch صورة أو يجعلها فى المركز . اضغط الأزرار المناسبة فى ال editor لأداء



## الباب السابع : إنشاء الـ Toolbars ، الـ Coolbars ، والـ Status Panels

هذه الأعمال . ولأن الأيقونات ثابتة من ناحية الحجم واللون ، فهذه الخيارات تعتبر مبטلة بالنسبة للأيقونات .

راجع شكل (٧-١٣) . لاحظ أن كل أيقونة قد تم تعيين قيمة صحيحة مفهرسة لها ، بدءاً من الصفرة للصورة الأولى . استخدم قيم الفهرس هذه لتعيين خاصية لأزرار الـ ToolBar . وهناك خطوتان لأداء هذا :

١- اختر الـ ToolBar وعين ImageList object لخاصية الـ Images . هذا يربط الـ ToolBar بمجموعة من الصور- أيقونات الزر في هذا المثال .

٢- اختر كل زر منفرد في الـ ToolBar ، وعين لخاصية الـ ImageIndex الخاصة بالزر قيمة الفهرس للصورة التي تريدها .

هذا هو الحد الأدنى من الخطوات المطلوبة لعرض صور أيقونة على أزرار الـ ToolBar . ولكن قد تريد أن تستخدم طريقتين أخريتين لعرض صور الأيقونة . قد تفعل هذا لتوفر صور مختلفة عندما تكون الأزرار مبטلة ، وعندما يحرك المستخدم مؤشر الفأرة فوق الزر . إن استخدام صور مختلفة (أو نفس الصور لكن بألوان مختلفة) .

يستخدم الـ CoolDemo واحداً من هذه الأنماط لتلوين أزرار عندما يشير مؤشر الفأرة إليها . لترى التأثير ، قم بتشغيل البرنامج مرة أخرى وحرك الفأرة فوق الأزرار الأربعة . (اضغط الزر الخاو الكبير لتشغيل زر الـ ToolBar الأولين) . كما ترى ، يبدو كل زر مضيقاً عندما تمرر مؤشر الفأرة فوق صورته . يدرك غالبية المستخدمين هذا الدليل البصري كمشير إلى أن الزر " جاهز " للضغط .

لإنشاء هذا التأثير في الـ CoolDemo ، فقد أنشأت ImageList آخر يسمى NavigatorHotImages (في الحقيقة هذا هو نفس الـ object الموجود في برنامج عرض متصفح الـ Web في Delphi) . والـ ImageList هذا يحمل الأيقونات المتطابقة كالقائمة الأخرى ، ولكن الصور قد تم تلوينها بالألوان الزاهية . إن تعيين الـ ImageList الخاصية الـ HotImages بالـ ToolBar يسبب استخدام هذه الأيقونات عندما تتحرك مؤشر الفأرة فوق زر في وقت التشغيل .

والطريقة الثالثة لاستخدام الـ ImageList في الـ ToolBar هي بتعيين مجموعة ثالثة من الأيقونات لخاصية الـ DisabledImages . إنني لا أفعل هذا في

ال CoolDemo لأن الأيقونات المبطله حسب النظام الافتراضى - المأخوذة من ال ImageList ومعينة لخاصية ال Images ومعتمة تلقائياً- تعتبر كافية فى الغالب . ولكن ، يمكنك تعيين DisabledImages اذا أردت استخدام صبور معينة لأزرار ذات خاصية Enabled محددة بـ False . يمكنك تحريك أيقونة- لإغلاق باب ، مثلاً أو عرض نوع آخر من الصور- عندما يكون الزر مبطل .

**فكرة:** لاستخدام الايقونات الموضحة فى تطبيق ال CoolDemo لهذا الباب ، فقط انسخ (اختر واضغط Ctrl+C) ال NavigatorImages وال NavigatorHotImages واحداً فى كل مرة فى نافذة ال form الخاص تطبيقك (اضغط Ctrl+V) .

### ال Coolbar Animations

من ابسط الاشياء إضافة ايقونة متحركة لل Coolbar ، والتي تشير الى ان البرنامج يعمل ، وكذلك إضافة flashy graphics للعرض . يعرض تطبيق ال CoolDemo قرص "Cool" دائرة ، والذي اقترضته من برنامج عرض متصفح Web الخاص بـ Delphi . والصورة المتحركة هى ملف واجهة تطبيق قياسى (.avi) ، تم تخزينه فى ملف Source\Data\Cool.avi (وايضاً متوفر مع Delphi فى دليل Demo\CoolStuff)

من السهل إضافة صورة متحركة لل Coolbar . ببساطة اسقط Animate component من ال Win32 palette على ال Coolbar . تأكد من انك اسقطت ال Animate على ال Coolbar ، وليس ال ToolBar . فهذا يجعل من السهل وضع ال Animate فى المكان الذى تريده . (ولكن يمكن ان يحمل ال ToolBar ال Animate) .

ان إسقاط ال Animate على ال Coolbar ينشئ Band جديد للصور المتحركة . فى الغالب ستريد تحريك هذا ال band الى موضع جديد- مثلاً ، يعرض ال CoolDemo ايقونة المتحركة الى اليمين من ال ToolBar الخاص به . لتحريك ال Band الصور المتحركة :

١ - اختر Animate object وحدد خاصية ال AutoSize له بـ False .

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

٢- ثم، مازلنا داخل Delphi، اسحب القضيب الرأسى للـ band الى اقصى اليسار، واسقط الـ Band الى اليمين من الـ ToolBar، أو فى أى مكان آخر تريده. قد تجرب عدة محاولات حتى تحصل على التأثير الذى تريده- من الافضل حفظ المشروع قبل البداية تشكيل الـ Coolbar bands.

٣- عين إسم ملف الـ avi. لخاصية الـ FileName الـ Animated.

٤- حدد خاصية الـ Active بـ True ليبدأ التحرك، والذى يتكرر تلقائياً.

لبدء إيقاف الصور المتحركة فى وقت التشغيل، عين True أو False لخاصية الـ Active. يفعل الـ CoolDemo هذا فى OnClick event handler باستخدام هذه العبارات:

```
Animate1.Active := not Animate1.Active;
```

```
if Animate1.Active
```

```
then Button1.Caption := 'Stop Animation'
```

```
else Button1.Caption := 'Begin Animation';
```

يحدد هذا الـ code خاصية الـ Active للحالة العكسية من الـ True أو الـ False لها، ثم تغير تعليق الزر بالنسبة لها.

**ملحوظة:** يضع الـ Windows حداً لحجم صورة AVI التى يمكن ان يعرضها الـ Animate control، وبذلك، تنطبق نفس هذه الحصول على الـ Animate لـ Delphi الذى يتوصل لـ Windows control. على سبيل المثال، يجب ان يكون حجم عرض الصورة اقل من 64k ولا يمكن ضغط الملف. ويمكن ان تنطبق حدود اخرى اعتماداً على التشكيل والطول الكلى للملف الـ AVI. إذن لا تكن مندهشاً اذا كان الـ Animate control الخاص بك غير قادراً على تشغيل كل ملف AVI تجده.

### Coolbar control اخرى:

كما ذكرت، يمكنك إضافة أى control للـ Coolbar band. والقاعدة الوحيدة هى الـ component يجب ان يكون من class مشتق من الـ TWinControl من أى صلة. على سبيل المثال، يمكنك إضافة الـ ComboBox للـ



Coolbar لأن الـ TComboBox class تشترك من الـ TComboBox، والتي بدورها تشترك من الـ TWinControl. ولكن، لا يمكنك إضافة TShape object للـ Coolbar لأن الـ TShape class الـ TWinControl تشترك من الـ TWinControl.

**ملحوظة:** يمكن من الناحية الفنية إسقاط non windowed object control على Coolbar، ولكن هذا لا يؤدي إلى إنشاء band جديد للـ control. للحصول على أفضل نتائج، ضع حداً للـ controls التي تضيفها للـ Coolbar إلى classes components مشتقة من الـ TWinControl.

Note

لإظهار كيف يمكن أن يحمل الـ Coolbar أى windowed control فقد أسقطت DateTimePicker من الـ Win32 palette على الـ Coolbar بالـ CoolDemo. راجع شكل (٧-٩)، أو قم بتشغيل برنامج الـ CoolDemo، لترى هذا الـ control. لأن الـ DateTimePicker يستقر على Coolbar، يمكنك للمستخدمين إعادة تحديد حجم الـ control وتحريكه إلى مكان آخر بضغط وسحب القضيبة الرأسى للـ band إلى أقصى اليسار.

لعرض الـ label "Date selector" إلى اليسار من الـ DateTimePicker، فقد فتحت الـ Bands Editor (بضغط الزر البيضاوى الواقع بعد خاصية الـ Bands للـ Coolbar)، واخترت الـ band الذى يحمل الـ TDateTimePicker. ثم بعد ذلك ادخلت الـ label فى خاصية الـ band object's Text وعلى عكس كثير من الـ components الأخرى ذات الـ labels، يجب أن تستخدم خاصية Text - الـ TCoolBand objects ليس لديها حقل Caption.

**ملحوظة:** لعرض الـ labels فى الـ Coolbar bands، يجب أن تكون خاصية الـ ShowText التابعة للـ Coolbar محددة بـ True.

Note

## الـ StatusBars :

كما أوضحت سلفاً فى هذا الباب، أن إحدى طرق إنشاء status panel بإضافة Panel object على الـ form. وتعطيك هذه التقنية تنوع واسع من خيارات العرض، ولكن هناك طريقة أبسط للحصول على status panels حسنة المظهر

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

باستخدام الـ StatusBar من الـ Win32 palette. في غالبية الحالات، يعتبر هذا الـ component هو كل ما تحتاج إليه لإنشاء اشربة المعلومات الـ status panels في نوافذ تطبيقك.

**ملحوظة:** لتفادي التداخل، إننى ادعو عنصر واجهة التطبيق بـ StatusBar عندما يستخدم الـ StatusBar والـ status panel عندما يستخدم الـ Panel الخاص بـ Delphi.

Note

### إنشاء الـ StatusBar:

تستقر الـ StatusBar عادة في اسفل النافذة، وهى تقدم معلومات متنوعة عن حالة البرنامج، سجل الـ StatusBar من الممكن أن ينقسم الى Panels (لا تخط بينها وبين الـ Panel)، كلاً منها يمكن ان يعرض نصاً أو صور جرافيكية. فى اغلب الحالات، يجب ان تعرض معلومات فى الـ StatusBar والتي يحتاج المستخدم ان يجدها بنظرة واحدة مثل ارقام العمود والخط، مواضع الفأرة، احجام الملف، ورسائل الخطأ. يمكنك ايضاً اختيارياً ان تضيف window gripper الى الركن الايمن السفلى من الـ StatusBar- هذا يجعل عملية إعادة تحديد حجم نافذة ذات حد رفيع اسهل.

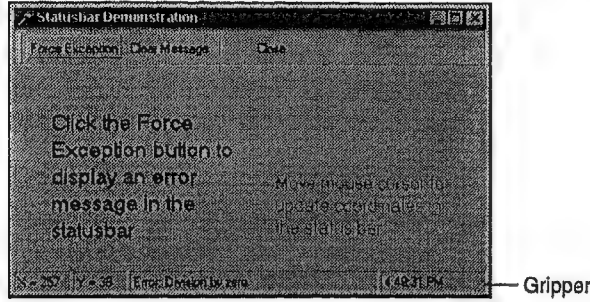
**على القرص المدمج:** يوضح شكل (٧-٤) مثال الـ statusbar من مشروع الـ Status، موجود على القرص المدمج المرفق فى دليل الـ Source\Status. لتشغيل البرنامج، ورؤية الـ source code، اكمل



الخطوات التالية:

- ١- افتح ملف مشروع Status.dpr واضغط F9.
- ٢- حرك مؤشر الفأرة داخل النافذة لتحديث قيم إحداثيات الـ X والـ Y المعروضة فى اللوحتين الأولىين للـ statusbar.
- ٣- اضغط زر الـ Force Exception لعرض رسالة خطأ فى منتصف الـ panel.
- ٤- اختر المستطيل المحيط لإعادة تحديد الحجم بالفأرة واسحبه لإعادة تحديد حجمه تلقائياً، ولكن لوحة الرسالة تغير الطول فقط.

تعمل كثير من هذه السمات في ظل سيطرة البرنامج، والذي كما سأوضح فيما بعد قائمة البرنامج، يعرض تقنيات عامة عديدة في برمجة ال StatusBar.



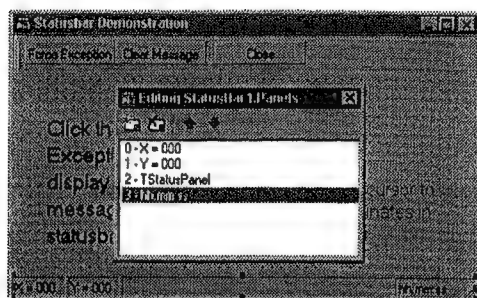
شكل (٧-١٤)، ينقسم ال StatusBarpanel الى لوحات، كلا منها يمكن ان يعرض نصاً أو صور جرافيكية. لاحظ المستطيل لإعادة تحديد حجم النافذة في الركن الايمن السفلى من StatusBar

لإنشاء StatusBar إسقط StatusBar من ال Win32 palette على form. في العادى، يقفز object الى اسفل النافذة، حيث يتم عرض غالبية StatusBar ولكن، كما هو الحال مع ال components الاخرى، يمكنك تغيير خاصية ال Align لوضع StatusBar على أى حافة من حواف النافذة (alRight، alLeft، alTop، أو alBottom)، أو ملئ ال client area بالنافذة (alClient). يمكن ان يكون لكل object من ال StatusBar object لوحة واحدة أو أكثر، كلا منها تعتبر object من ال TStatusPanel class. لإنشاء panels، إتبع هذه الخطوات:

- ١- اختر StatusBar فى نافذة ال form.
- ٢- اضغط الزر البيضاوى الواقع بعد خاصية ال Panels. هذا يفتح ال Status Panels Editor [انظر شكل (٧-١٥)].
- ٣- اضغط ايقونة ال Add New لإضافة Panel جديدة.
- ٤- اضغط Delete Selected لحذف اللوحة التى تم إبرازها. استخدم ازرار الاسهم المشيرة لأعلى وأسفل لتحريك اللوحة المختارة الى أعلى أو اسفل فى القائمة.

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

هذا المحرر هو الطريقة الوحيدة للوصول الى panel objects منفردة فى وقت التصميم . باستخدام المحرر، قم بإبراز اللوحة التى تريد تعديلها، ثم استخدم الـ Object Inspector لادخال قيم الخاصية .



شكل (٧-١٥): استخدام الـ Status Panels Editor لإنشاء، حذف، وترتيب panels للـ StatusBar

### خصائص الـ StatusBar

ان الـ StatusBar له خصائص عديدة يمكنك استخدامها لتشكيل الـ StatusBar بأكمله . تذكر ان كل panel فى الـ StatusBar التوصل إليها فقط من خلال الـ Status Panels Editor، الذى يفتح عندما تضغط الزر البيضاوى الواقع بعد خاصية الـ panel للـ StatusBar . وتتضمن الخصائص المختارة للـ StatusBar ما يلى :

● **Canvas** : استخدم هذا الـ object لتلوين النص والصور الجرافيكية فى الـ StatusBar panel . لا يجب عليك ان تفعل هذا الـ text panel النص فقط . استخدم هذا الحقل مع لوحات الرسم، التى سوف اشرحها بالتفصيل قرب نهاية هذا الباب (انظر لوحات الـ StatusBar للرسم) .

● **Panels** : هذه الخاصية هى array الـ TStatusPanel، كلاً منها يحدد مظهر ومحتوى StatusBar panel منفردة . فى وقت التشغيل، اضغط الزر البيضاوى الواقع بعد هذه الخاصية لفتح الـ Status Panels Editor . فى وقته التشغيل، يمكنك التوصل الى الـ StatusPanel objects المنفردة باستخدام هذا array - لتغيير نص panel's text، مثلاً .

• **SimplePanel**: إذا كنت تريد statusbar احادى ال Panel فقط ، حدد هذه الخاصية بـ True ودعك من اصطفااف ال Panel . استخدم الحقل التالى ، SimpleText ، لعرض رسالة فى ال statusbar . يمكنك أيضاً استخدام هذا الحقل لتحول مؤقتاً statusbar متعددة ال Panels الى واحد بسيط- لعرض رسالة طويلة، مثلاً. فقط ، حدد ال SimplePanel بـ True فى عبارة برنامج، وعين نص لل SimpleText . لإعادة statusbar إلى مظهرة المتعدد ال Panels ، حدد ال SimplePanel بـ False .

• **SimpleText**: عين أى string للعرض فى statusbar يكون حقل ال SimplePanel الخاص به محدد بـ True لعرض text فى statusbar متعدد ال Panels ، استخدم ال Panels array .

• **SizeGrip**: حدد هذه الخاصية بـ True لعرض المستطيل المحيط إعادة تحديد حجم نافذة اختياري فى الركن الايمن السفلى لـ statusbar . هذا يجعل من السهل على المستخدمين إعادة تحديد حجم النوافذ بالضغط والسحب بالفأرة .

### خصائص ال StatusPanel:

تعتبر كل Panel فى ال StatusBar ال object من ال TStatusPanel class . بالرغم من ان هذا يعد component ، الا أنه لا يظهر فى ال VCL palette . ان الطريقة الوحيدة لإنشاء وتعديل TStatusPanel هو باستخدام ال Status Panels Editor (اضغط الزر البيضوى الواقع يعد خاصية ال Panels لـ StatusBar object's) استخدم المحرر لإضافة ، حذف ، وإعادة ترتيب ال Panels . اختر Panels فردية ثم استخدم ال Object Inspector لتعديل الخصائص التالية :

• **Alignment**: تؤثر هذه الخاصية على كيفية عرض ال text فى ال panels . حدد هذه الخاصية بـ taLeftJustify (وهو التحديد الافتراضى) لعرض ال text flush على الحد الايسر؛ استخدم ال taLeftJustify لعرض نص على جانب الحد الايمن ، أو استخدم ال taCenter لتمركز ال text داخل عرض ال panel .

• **Bevel**: لل panels ثلاث خيارات عرض . استخدم ال pbLowered (وهو الافتراضى) للحصول على تأثير 3D ، pbRaised للحصول على panel مرتفعة ، أو pbNone للحصول على مظهر مسطح .

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

● **Style**، حسب المعتاد، حدد هذه الخاصية بـ psText حتى يقوم الـ Windows تلقائياً بتحديث محتوى الـ text داخل الـ panel. اذا حددتها بـ psOwnerDraw، فإنك مسؤول عن عرض محتوى الـ panel في الـ OnDrawPanel event handler الخاص بالـ StatusBar، باستخدام خاصية الـ Canvas للـ StatusBar. مؤخراً في هذا الباب، سوف اشرح كيفية استخدام النمط والحدث لعرض صور جرافيكية في الـ StatusBar panels (انظر "لوحات الـ StatusBar للرسم").

● **Text**، عين text string للعرض في الـ panel. يتم محاذاة النص classes لخاصية الـ Alignment، ويتم عرضه تلقائياً اذا كانت الـ Style محددة بـ psText. يمكنك استخدام هذه الخاصية لتخزين text panel اذا كانت للـ Style محددة بـ psOwnerDraw، ولكن في هذه الحالة انت مسئول عن عرض النص في الـ panel.

● **Width**، هذه عرض الـ panel بالـ pixels. بالرغم من ان الـ StatusBar يقوم باعادة تحديد الحجم تلقائياً عندما يتغير حجم النافذة، الا انك قد تريد التحكم في عروض الـ panels الفردية. فمثلاً، قم بتشغيل التطبيق Status واعد تحديد حجم النافذة- لاحظ ان الـ panels الوسطى الكبيرة فقط هي التي يتغير طولها، وتبقى الاخرى بنفس الحجم. وهذا يتطلب بعض البرمجة، كما سأوضح بعد قائمة البرنامج التالية.

### التطبيق Status:

على القرص المدمج: ان تطبيق Status، الموجود في دليل source\Status. على القرص المدمج المرفق بهذا الكتاب، يظهر عناصر رئيسية متعددة في برمجة الـ StatusBar توضح القائمة (٧-٧) الـ source code لبرنامج الـ Status. راجع شكل (٧-١٤) لرؤية عرض البرنامج.



القائمة (٧-٧): Status\Main.pas

```
unit Main;
```

```
interface
```

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, ComCtrls, StdCtrls, Buttons, ExtCtrls,  
ToolWin;

type

```
TMainForm = class(TForm)
  Label1: TLabel;
  StatusBar1: TStatusBar;
  Timer1: TTimer;
  Coolbar1: TCoolbar;
  ToolBar1: TToolBar;
  ToolButton1: TToolButton;
  ToolButton2: TToolButton;
  ToolButton3: TToolButton;
  ToolButton4: TToolButton;
  Label2: TLabel;
  procedure FormMouseMove(Sender: TObject;
    Shift: TShiftState; X, Y: Integer);
  procedure FormResize(Sender: TObject);
  procedure Timer1Timer(Sender: TObject);
  procedure ToolButton1Click(Sender: TObject);
  procedure ToolButton2Click(Sender: TObject);
  procedure ToolButton4Click(Sender: TObject);
  procedure Label1MouseMove(Sender: TObject;
    Shift: TShiftState; X, Y: Integer);
  private
  { Private declarations }
  public
  { Public declarations }
end;
```

## الباب السابع : إنشاء الـ Toolbars ، الـ Coolbars ، والـ Status Panels

var

MainForm: TMainForm;

implementation

{ \$R \*.DFM }

{ Rather than use literal index values, this section defines descriptive constants for the statusbar's four panels }

const

XPanelIndex = 0;

YPanelIndex = 1;

MessagePanelIndex = 2;

TimePanelIndex = 3;

{ Update X and Y coordinate values in the statusbar }

procedure TMainForm.FormMouseMove(Sender: TObject;  
Shift: TShiftState; X, Y: Integer);

begin

StatusBar1.Panels[XPanelIndex].Text := 'X = ' + IntToStr  
(X);

StatusBar1.Panels[YPanelIndex].Text := 'Y = ' + IntToStr(Y);

end;

{ Calculate width of middle panel so the others stay the  
same size when the window resizes. }

procedure TMainForm.FormResize(Sender: TObject);

const

Fudge = 25; // Allow extra space for width calculation

var

W: Integer; // Width of fixed-size panels

begin

with StatusBar1 do

begin



```

W := Panels[XPanelIndex].Width +
    Panels[YPanelIndex].Width + Panels[TimePanelIndex].Width;
    Panels[MessagePanelIndex].Width := Width - (W +
Fudge);
    end;
end;

{ Display the time in the rightmost statusbar panel }
procedure TMainForm.Timer1Timer(Sender: TObject);
begin
    StatusBar1.Panels[TimePanelIndex].Text := TimeToStr(Time);
end;

{ Force an exception to occur and display its message
in a statusbar panel. The call to ShowMessage is never
made, but is included to prevent the compiler from
optimizing out the integer division. }
procedure TMainForm.ToolButton1Click(Sender: TObject);
var
    K, J: Integer;
begin
    K := 100; J := 0;
    try
        K := K div J; // Force divide by zero exception
        ShowMessage(IntToStr(K) + ':' + IntToStr(J));
    except on E: Exception do
        begin
            MessageBeep(0);
            StatusBar1.Panels[MessagePanelIndex].Text :=
                'Error: ' + E.Message;
        end;
    end;
end;
end;

```

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

```
{ Clear the text from middle statusbar panel }
procedure TMainForm.ToolButton2Click(Sender: TObject);
begin
    StatusBar1.Panels[MessagePanelIndex].Text := "";
end;
```

```
{ End the program }
procedure TMainForm.ToolButton4Click(Sender: TObject);
begin
    Close;
end;
```

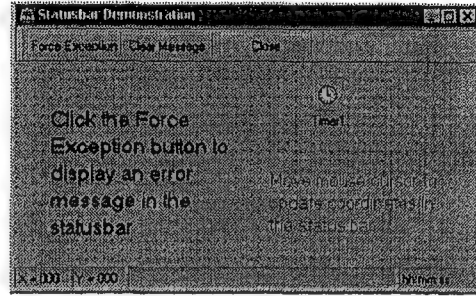
```
{ Translate mouse coordinates when the cursor moves over
one of the two large labels, and pass the results on
to the MainForm's OnMouseMove event handler. }
procedure TMainForm.Label1MouseMove(Sender: TObject;
Shift: TShiftState; X, Y: Integer);
```

```
var
    T: TLabel; // Refers to Sender as a TLabel object
begin
    T := Sender as TLabel; // Initialize T
    // Pass coordinates to MainForm's event handler
    MainForm.FormMouseMove(Sender, Shift, T.Left + X,
    T.Top + Y);
end;
```

end.

يعرض برنامج الـ four-panel statusbar في اسفل النافذة. لإنشاء الـ Panels، فقد قمت بفتح الـ Status Panels Editor (بضغط الزر البيضاوي الواقع بعد خاصية الـ Panels للـ StatusBar) واضفت أربع panel objects. لقد حددت أولاً عرض الـ Panel الثالثة بـ 250، تاركاً الآخرين محددين بالعرض الافتراضي وهو 50. وكذلك ادخلت نصاً في كل الـ Panels الا الـ Panels

الكبرى ، حتى تعرض شيئاً فى وقت التصحيح يذكرنى بالغرض من الـ Panel. هذا ليس ضرورياً رغم ذلك. يوضح شكل (٧-١٦) مظهر الـ form فى وقت التصميم. تكون ايقونة الـ Timer غير مرئية فى وقت التشغيل هذه الـ object توفر الـ Timer event الذى يستخدمه البرنامج لتحديث الوقت فى اقصى يمين الـ Panel.



شكل (٧-١٦): نافذة برنامج الـ Status فى وقت التصميم

لأغراض العرض ، تعرض الـ two panels الأولتان فى الـ StatusBar الخاص بالبرنامج إحدائيات X و Y الخاصة بمؤشر الفأرة. قم بتشغيل البرنامج وحرك مؤشر الفأرة داخل نافذة البرنامج- الإحدائيات موضحة فى الـ StatusBar. هناك Two procedures لتحديث هذه المعلومة. الأول OnMouseMove event handler لـ MainForm object's ، والذى ينفذ عبارات :

```
StatusBar1.Panels[XPanelIndex].Text := 'X = ' + IntToStr(X);
```

```
StatusBar1.Panels[YPanelIndex].Text := 'Y = ' + IntToStr(Y);
```

هذا يوضح كيفية التوصل الى panels منفصل فى الـ StatusBar باستخدام الـ Panels array ، يمكن ان تكون قيم الفهرس الموجودة فيما بين الاقواس اعداد صحيحة حرفية (ان فهرس الـ Panel الأولى يكون صفر دائماً) ، ولكننى قمت بتحديد ثوابت واصفة مثل XPanelIndex (انظر قطاع الـ const فى منتصف القائمة تقريباً). لانك قد تضيف وتغير ، أو تعيد ترتيب الـ panels ، اثناء تصميم برنامجك ، فمن الصواب ان تستخدم ثوابت واصفة بدلاً من اعداد صحيحة حرفية. اذا قمت بتغيير الـ StatusBar's panels ، يمكنك ببساطة تحديث تعيينات الثوابت تبعاً لها.

## الباب السابع :إنشاء ال Toolbars، ال Coolbars، وال Status Panels

ان إثنان من ال text objects الكبيرين بالبرنامج، الذان يخبرانك ان تضغط زر ال Force Exception وان تحرك مؤشر الفأرة، يتلقيان events تحرك الفأرة الخاصة بهم. لهذا السبب، يعتبر ال procedure الثانى ضرورياً لتحديث قيم إحداثيات ال X وال Y فى ال StatusBar عندما تمر بمؤشر الفأرة فوق النص. وهذا يعتبر محيراً بعض الشيء، ويتم تنفيذه فى ال LabelMouseMove event procedure، والملاحظ هو الآخر بـ OnMouseMove event 2 label و label 1 objects لها نظام الاحداثيات الخاص بها، وهو [0,0] فى الركن الايمن العلوى. لعرض موضع الفأرة بالنسبة للـ client area بالنافذة، من الضروري ان تضيف موضع كل label [Left, Top] للاحداثيات التى يتقبلها ال OnMouseMove event. labels.

ان تحديث الوقت فى ال StatusBar panel يعد أمراً بسيطاً. لفعل هذا، قمت بإضافة ال Timer object على ال form، وقمت ببرمجة ال event handler الخاص لها ال OnTimer لتنفيذ عبارة:

```
StatusBar1.Panels[TimePanelIndex].Text :=  
TimeToStr(Time);
```

مرة اخرى، ان استخدام ثابت ال TimePanelIndex يعتبر أكثر أمناً ووصفاً من قيمة فهرس عدد صحيح حرفية للتوصل ال ال Panels array. ان تحديد خاصية ال Text لهذه ال panel هو كل ما تحتاجه لعرض الوقت كل ثانية، وهو التردد الافتراضى ال ال Timer's event.

لمحاكاة عرض رسالة خطأ- وهى احدى الاستخدامات النموذجية للـ StatusBar panel- يقوم ال ToolButton1Click بقسمة متغير عدد صحيح على صفر. هذا يجعل جزئية ال except فى قالب ال try-except تحدد صغيراً وتعرض حقل ال Message الخاص بال exception باستخدام ال code:

```
MessageBeep(0);  
StatusBar1.Panels[MessagePanelIndex].Text :=  
'Error: ' + E.Message;
```

مرة اخرى، لقد استخدمت ثابت واصف للتوصل الى ال StatusBar panel الصحيحة فى ال Panels array ان تحديد خاصية ال Text يعرض رسالة خطأ.

لمسح هذه الرسالة، يعتبر ToolButton event handler آخر null string لنفس هذه الخاصية:

```
StatusBar1.Panels[MessagePanelIndex].Text := ";
```

**ملحوظة:** عندما تقوم بتشغيل برنامج من داخل Delphi، إذا حدث exception، يعرض مزيل الاخطاء في Delphi حالة البرنامج الحالي وقد يفتح نافذة CPU. إذا حدث هذا، اختر Tools\Environment Options، اضغط على Debugger page tab، وقم بإبراز ال Exceptions الخاصة بـ Delphi في مربع القائمة Exceptions لمنع مقاطعة البرنامج وتشغيله للتعامل مع ال exception بشكل طبيعي، حدد زر ال Handled By الى User Program. للعودة الى مقاطعة البرنامج عند تلقي exception، قم بتغيير هذا الزر مرة أخرى الى Debugger.

عندما تعيد تحديد حجم نافذة برنامج Status، يقوم ال StatusBar تلقائياً بإعادة تحديد حجمه ليتلائم مع الحافة السفلية للنافذة. ولكن، هذا التأثير الافتراضي ليس كافياً للتوصل الى ما أردته - أن أجعل لوحات الوقت والإحداثيات تبقى ثابتة من ناحية الحجم، وأن أجعل ال panel الكبير فقط تكبر أو تصغر حسب الحاجة. إن الحصول على هذا التأثير يستلزم بعض البرمجة. لترى هذا، قم بتشغيل Status واحد تحديد حجم النافذة. اختر كيف تتغير ال StatusBar panel الثالثة فقط في الطول.

لتوفير مكان لل code التي تتحكم في عروض ال panel يستخدم Status ال OnResize لنافذة البرنامج. يتم استدعاء هذا ال event عندما يتم عرض النافذة لأول مرة، وكذلك في أي وقت يتغير فيه حجم النافذة. لتحديث عروض اللوحة، ينفذ ال event handler هذا ال code:

```
with StatusBar1 do
begin
  W := Panels[XPanelIndex].Width +
    Panels[YPanelIndex].Width + Panels[TimePanelIndex].Width;
  Panels[MessagePanelIndex].Width := Width - (W + Fudge);
end;
```

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

إن متغير W محدد بإجمالي عروض الـ panels الثلاث ذات الحجم الثابت. وهذه القيمة، بالإضافة الى عامل الـ "fudge factor"، يتم طرحها من إجمالي عرض الـ StatusBar. والنتيجة تساوى العرض المنشور للـ message panel، والتي يتم تعيينها لخاصية الـ width لهذه الـ panel. وكنتيجة لهذا الـ code، فإن إعادة تحديد حجم النافذة يتغير فقط طول الـ panel الثالثة. وتبقى الـ panels الوقت والإحداثيات ثابتة الحجم.

### :Owner-draw StatusBar panels

عندما تحتاج مزيداً من التحكم فى كيفية عرض العناصر فى StatusBar panel، يمكنك الإشارة الى أن panel تحتاج الى تحديث. قد تستغل هذه الفرصة لرسم صور جرافيكية على الـ panels أو لعرض نص باستخدام fonts مختلفة. هذا يسمى StatusBar panel للـ owner-draw (أو، بتعبير أكثر شمولاً، owner-draw control).

فى العادى، يعتنى الـ Windows بعرض نص فى StatusBar panels. كل ما تحتاج أن تفعله هو أن تعين هذا النص، مثلاً، فى event handler. قم بتعيين أى string لخاصية الـ panel object's Text باستخدام عبارة مثل التالية :

```
StatusBar1.Panels[2].Text := 'Insert mode';
```

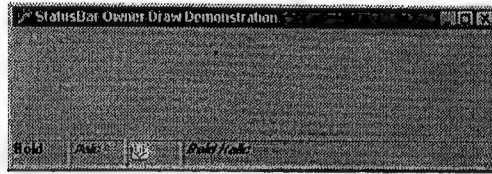
ان تعيين أى string لخاصية الـ Text للـ panel يعرض على الفور هذا النص. اذا احتجت بعد ذلك الى تحديث بالـ panel مثلاً، عندما يتغير حجم نافذة أو عندما يتم كشفها من قبل نافذة اخرى يقوم الـ Windows تلقائياً بإعادة عرض نص الـ StatusBar panel باستخدام الـ string الذى تم حفظه فى خاصية الـ Text.

لتولى هذه المسئولية، حدد خاصية الـ Status للـ panel بـ psOwnerDraw باستخدام الـ Status Panels Editor (اضغط الزر البيضاوى الواقع بعد خاصية Panels للـ StatusBar). يمكنك فعل هذا مع الـ Panels المختارة، أو معهم جميعاً. مع اختيار هذا النمط، فمتى يطلب الـ Windows ان تعرض الـ Panels محتوياتها، يتم اطلاق الـ OnDrawPanel event للـ StatusBar object. وهذا الـ procedure، يمكنك إدخال عبارات البرمجة التى احتاجها لعرض عناصر فى الـ StatusBar panel.

**ملحوظة:** ان ال OnDrawPanel event مرتبط ال StatusBar StatusOD object . فكل panel فى ذلك ال object يجب ان تكون خاصية ال Style لها محددة بـ psOwnerDraw حتى يتم إطلاق ال event لهذه ال panel .



**على القرص المدمج:** المثال هنا لمعرفة كيفية فعل هذا، وهو مشروع ال StatusOD ( OD تعنى "Owner Draw" ) الذى يعرض كيفية تغيير انماط ال fonts لبنود النص فى ال StatusBar panels ، وايضاً كيفية رسم صورة ايقونة داخل panel . يوضح شكل (٧-١٧) عرض البرنامج . توضح القائمة (٧-٨) ال source code للبرنامج . توجد ملفات البرنامج على القرص المدمج المرفق بهذا الكتاب فى دليل ال Source\StatusOD . لتحميل هذا البرنامج فى Delphi ، افتح ملف StatusOD.dpr ، واضغط F9 التشغيل .



شكل (٧-١٧): ال Owner-draw StatusBar panels تجعل من الممكن تغيير ال text fonts وعرض صور جرافيكية كما يوضح برنامج StatusOD

القائمة (٧-٨): StatusOD\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, Images, ComCtrls;
```

```
type
```

```
TMainForm = class(TForm)
```

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

```
StatusBar1: TStatusBar;
ImageList1: TImageList;
procedure StatusBar1DrawPanel(StatusBar: TStatusBar;
    Panel: TStatusPanel; const Rect: TRect);
private
{ Private declarations }
public
{ Public declarations }
end;
var
    MainForm: TMainForm;
implementation
{$R *.DFM}
procedure TMainForm.StatusBar1DrawPanel(StatusBar: TStatusBar;
    Panel: TStatusPanel; const Rect: TRect);
var
    SavedStyles: TFontStyles;
begin
    with StatusBar, Canvas do
        begin
            SavedStyles := Font.Style; // Save current font styles
            // Set font style or draw an icon depending
            // on which Panel needs updating.
            case Panel.Index of
                0 : Font.Style := [fsBold];
                1 : Font.Style := [fsItalic];
                2 : ImageList1.Draw(Canvas,
                    Rect.Left + 2, Rect.Top + 2, 0);
                3 : Font.Style := [fsBold, fsItalic];
            end;
            // Draw text in all panels except the third,
            // which displays an icon.
            if Panel.Index <> 2 then
```



```

TextRect(Rect, Rect.Left, Rect.Top, Panel.Text);
Font.Style := SavedStyles; // Restore saved font styles
end;

end;

end.

```

يوضح الـ `StatusBar1DrawPanel Procedure` كيفية برمجة الـ `StatusBar panels owner-draw`. كل `panel` فى برنامج العرض له خاصية الـ `OnDrawPanel` محددة بـ `psOwnerDraw`. هذا يؤدى الى ان يتلقى `OnDrawPanel` event handler الخاص بالـ `StatusBar` إشارة عندما تحتاج الـ `panel` الى إعادة الرسم. قد يكون هذا عندما يبدأ البرنامج لأول مرة، وعندما تغير النافذة حجمها، أو عندما تقوم عبارة أخرى فى مكان ما بالبرنامج بتعيين `string` جديد لخاصية الـ `Text` للـ `panel` فى جميع الحالات، يتم استدعاء `OnDrawPanel` لتحديث محتويات الـ `panel`. فى هذا المثال، لان الـ `StatusBar` له اربعة `panels`، يتم استدعاء `OnDrawPanel` اربع مرات: مرة لكل `panel`، متى يطلب الـ `StatusBar` تحديثاً.

واحدة من أولى مهام الـ `event handler` هى تحديد أى من الـ `panel's` التى تحتاج إعادة رسم ما عليها. ان اسهل طريقة لفعل هذا هى باستخدام خاصية الـ `Index` للـ `Panel parameter`. هذا يساوى قيمة الـ `index` التى تصل الى الـ `panel object` فى الـ `Panels array` فى الـ `StatusBar`. قد يكون افضل اسلوب لفهم هذه العلاقة من الناحية البرمجية - العبارة التالية صحيحة:

```
StatusBar.Panels[Panel.Index] = Panel;
```

الـ `event handler` يتلقى الـ `StatusBar parameter`. والذى يمثل الـ `object` الذى يحتوى على الـ `Panel`. إن الـ `Panel object` يوجد مرجعه فى الـ `Panel parameter`. وهناك `parameter` ثالث، وهو `Rect`، يعطى أبعاد الـ `panel` على الشاشة - اذا كانت النافذة تغير حجمها، فإن هذا السجل يحتوى على الحجم الجديد للـ `panel`.

لرسم نص بأنماط مختلفة، يعين البرنامج مجموعات من ثوابت النمط. لخاصية الـ `Font` التابعة لـ `StatusBar.Canvas`. إننى أقفز هنا بعض الشئ، إن

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

مفهوم الـ Canvas سيتم شرحه في الباب الثالث عشر ، وهو تطوير تطبيقات الجرافيك . تخيل الـ Canvas على أنه سطح تستطيع أن ترسم عليه الـ control في حالة الـ StatusBar ، يقدم الـ Canvas السطح الذي ترسم عليه أو تلون نصاً داخل panel منفردة .

تقوم عبارة الـ with في الـ procedure بتبسيط بعض الـ code . على سبيل المثال ، لتعيين نمط الخط الـ fsBold لخاصية الـ Font.Style للـ Canvas ، فإن عبارة الحالة تنفذ العبارة :

```
Font.Style := [fsBold];
```

إن أنماط الخط تعتبر مجموعات ، ولذا ، فبالرغم من أن هناك قيمة واحدة فقط محددة ، فيجب أن تكون بين أقواس . إذا لم يتم استخدام عبارة الـ with ، فلن تكتب هذه العبارة :

```
StatusBar.Canvas.Font.Style := [fsBold];
```

كلتا العبارتين متساويتان ، ولكن عبارة الـ with تبسط العبارة الأولى باخبار الـ Delphi أن يستخدم خاصية الـ Font في الـ Canvas الخاص بالـ StatusBar وهذا حسب النظام الافتراضي . بعد تحديد نمط الخط ، يستدعي البرنامج الـ Canvas method التابع للـ StatusBar لعرض نص باستخدام هذا النمط . وهذه العبارة :

```
TextRect(Rect, Rect.Left, Rect.Top, Panel.Text);
```

ترسم قيمة الـ Text الخاص بـ Panel object باستخدام الـ Rect وقد تم تمريره كـ parameter event handler . ان اول ذكر للـ Rect هنا يحدد الحدود لقدرة الانتاج - اذا زاد رسم النص عن حجم الـ panel ، فيتم قصه ليتناسب . وتحدد القيمتين الجديدتين موضع الركن الايسر العلوي لبند النص . والـ parameter الاخير بالطبع هو الـ string الذي سوف يرسم .

ان رسم الصور الجرافيكية في panel يتم بنفس السهولة . لفعل هذا في البرنامج ، تستدعي جملة الـ Draw method في الـ ImageList object الخاص بالبرنامج :

```
ImageList1.Draw(Canvas,  
Rect.Left + 2, Rect.Top + 2, 0);
```

لا يجب عليك ان تستخدم الـ ImageList object ، ولكنها ابسط طريقة لادخال الـ bitmaps ، وبخاصة فى حالة الايقونات . يقوم الـ ImageList component بفعالية بتخزين مثل هذه الصور لـ bitmaps فردية فى الذاكرة . قم بتحديد رقم الـ index للصورة التى تريدها (وهو صفر فى هذه الحالة ، وقد تم تمريره كأكبر قيمة متغيرة للـ Draw) ، ويسترجع الـ ImageList object هذه الصورة . لاحظ كيف تحدد عبارة الـ Draw الـ Canvas الخاص بـ StatusBar1 الـ object .

### افكار للمستخدم الخبير

● عين قيمة الخاصية الـ HintPause للـ application object لتغيير تأخير الوقت ليظهر مربع hint box . ان القيمة الافتراضية هى ٨٠٠ على ثانية . على سبيل المثال ، لبرمجة ظهور اسرع للـ hint box ، يمكنك تعيين ٢٥٠ (أى ١/٤ ثانية) للـ Application.HintPause . قد تفعل هذا فى أى event handler (استخدم الـ OnCreate للـ form لتوصيف الـ hints بمجرد بدء التشغيل) .

● ان لون الـ hint box اصفر والنص بها اسود ، وهو ما يمثل الـ Post-It® Notes الاصلية . لتغيير هذا اللون ، يمكنك تعيين قيمة الـ TColor لخاصية الـ HintColor للتطبيق . فمثلاً ، لإنشاء hint box حمراء ، عين الـ clRed للـ Application.HintColor . (انظر نوع بيانات الـ TColor فى الـ online help الخاصة بـ Delphi لمعرفة المزيد من المعلومات) تذكر ان المستخدمين باستطاعتهم ان يغيروا لون الـ hint box بتعديل تحديدات نظام الـ Windows .

● عند تصميم الـ status panels مقسمة وتملك أكثر من الـ TPanel component ، اختر font للـ panel الرئيسية ، ثم حدد خاصية الـ ParentFont بـ True لكل panel فرعية . عندئذ ، يستخدم كل تقسيم فرعى تحديدات بنط الـ panel الرئيسية . فهذا اسهل من تعيين خصائص البنط لكل panel فرعية .

## الباب السابع : إنشاء ال Toolbars ، ال Coolbars ، وال Status Panels

● يمكنك استخدام TPanel objects كسطح مرتفعة لعرض نص في أى مكان من نافذة ال form- لا يجب عليك قصر ال TPanels على الاستخدام ل toolbar و status panel على سبيل المثال ، يستخدم ال Compiling information dialog الخاص بـ Delphi ال TPanels لعرض المعلومات أثناء ال Compiling . لرؤية هذه النافذة ، إختار امر ال Options|Environment ، إختار باب ال Preferences page tab لعرض ال compiler progress وقم بإجراء عملية ال compile لمثال .

● أضف Panel على ال form وحدد خاصية ال Align لها بـ alClient . هذا يجعل ال Pane تملأ نافذة ال form تماماً . إختار قيم لل BevelInner وال BevelOuter ، وقم بتحديد أكثر من قيمة لل BevelWidth وال BorderWidth (١٠ و ٢٥ ، مثلاً) . يمكنك الحصول خلفيات نافذة ممتعة باستخدام ال Panel كخلفية نافذة . (إذا أضفت ال components ، إختار ال Panel وإختار (Edit|Send to Back.

● بدلاً من استخدام ال glyph bitmap ، يمكنك استخدام خاصية Font لل SpeedButton لعرض رموز . حدد ال Font الخاص بالزر بينط ال TrueType مثل ال WingDings . لا يزال ال Caption يعرض رموز نص ASCII . هذه التقنية تعمل مع أى عنصر نص له خاصية Font .

● عند استخدام ال Format function ، فالخطأ الشائع هو ان تنسى إدخال ال arguments فى الاقواس . تذكر ان Args parameter الخاص بال Format يعد array- بعبارة اخرى ، قائمة parameter متغيرة الاطوال . على سبيل المثال ، يمرر التعبير [V1,V2,V3] لل Args المتغيرات V1 ، V2 ، و V3 .

● ان كلاً من TCoolbarclasses و ال TToolBar classes تنحدر ، من TToolWindow classes ، والتي تحدد السمات لنافذة ذات حدود و client area مخصصة . هذا يجعل ال TCoolbar و ال TToolBar اقارب (يتشاركون فى نفس ال classes الأم) . لمعرفة المزيد عن ال TCoolbar components و ال TToolBar ، قد تريد ان تتعرف على ال online والتوثيقات المطبوعة لل TToolWindow class .

• ال Coolbar عبارة عن windowed control ، وقد يتم إدخاله فى Coolbar آخر لإنشاء multilevel concoction . قد تستخدم هذه التقنية لإنشاء Coolbars ذات bands مرتبة رأسياً وأخرى مرتبة أفقياً- فى الواقع ، هذه هى الطريقة الوحيدة للحصول على هذا التأثير . ولكن ، تأكد من أنك فعلاً تحتاج الى هذه التعقيدات الخاصة بال multilevel Coolbar قبل ان تبذل مجهوداً فى تنفيذ الفكرة .



### المشروعات التى يمكنك تجربتها

(١-٧): قم بإنشاء toolbar يمكن للمستخدمين تحريكه الى الحد العلوى للنافذة ، أو الحد السفلى ، أو الايمن ، أو الايسر بالإضافة الى تعيين قيمة جديدة لخاصية ال Align لل Panel ، يجب عليك ان تعدل خصائص ال Left وال Right لل SpeedButton الى مواضع جديدة قريبة داخل حدود ال Panel object بـ toolbar .

(٢-٧): قم بإنشاء ال status panel توضح التاريخ والوقت . (ملحوظة: راجع ال source code لتطبيق ال Toolbar2) . قم باختبار تطبيقك بتشغيله طوال الليل للتأكد من انه يغير التاريخ فى منتصف الليل .

(٣-٧): قم بتصميم floating toolbar لاختيار اللون ، وذلك باستخدام تطبيق ال Toolbar3 كمرشد لك .

(٤-٧): قم بتجربة الضغط والسحب لل objects مثل ال Buttons وال SpeedButtons فى وقت التشغيل . استخدم برمجة شبيهه بتلك الموجودة فى Toolbar3 للاستجابة لل events وال OnMouseDown ، وال OnMouseMove ، وال OnMouseUp .

(٥-٧): اكتب برنامج اختبار مع ال status panel التى تعرض رسالة خطأ . استخدم ال Format function لإنشاء integer error

## الباب السابع : إنشاء الـ Toolbars، الـ Coolbars، والـ Status Panels

code . اضع ازرار واوامر قائمة للبرنامج لمحاكاة انواع مختلفة من الاخفاء .

(٦-٧): ادخل التاريخ والوقت في الـ status panel .

(٧-٧): قم بتصميم form template مع الـ status panel من التطبيق CapsLock [راجع شكل (٧-٨) و القائمة (٧-٥)].

(٨-٧): قم بتحويل برنامج الـ Tabs الموجود على القرص المدمج في دليل الـ Source\Tabs ليستخدم الـ StatusBar objects والـ Win32 ToolBar الخاصة بالـ Win32 .

### ملخص:

- استخدم الـ Panel component لإنشاء toolbars و status panels .  
تحتوى الـ toolbars على SpeedButton objects . تعرض Status panels نصاً .

- قم بتجميع الـ StatusBar objects في مجموعات بتحديد خصائص الـ GroupIndex لها بنفس القيمة العددية الصحيحة الموجبة غير الصفرية . تعمل الـ SpeedButton بعد ذلك مثل الـ RadioButtons- يمكن ان يكون زرأ واحداً فقط فى المجموعة الى اسفل . حدد الـ AllowAllUp بـ True لتسمح لجميع الازرار بالمجموعة ان تبقى فى حالة إغلاق .

- قم بتصميم floating toolbars بإنشاء الـ OnMouseDown ، OnMouseMove ، و OnMouseUp . قم باستدعاء الـ SetCapture function الخاصة بالـ Windows لتبدأ عملية ضغط وسحب . دائماً اجعل كل استدعاء للـ SetCapture ملحق باستدعاء للـ ReleaseCapture . انظر الباب الثانى عشر لمزيد من المعلومات عن امكانيات نافذة الوصل الجديدة والتى يمكنك استخدامها ايضاً لإنشاء floating toolbars .

- هناك تشابه بين الـ status panel و toolbars ولكنها غالباً ما تعرض نصاً بدلاً من الازرار . قم بتقسيم status panel بادخال الـ Panel objects فى الـ Panel الرئيسية . قم بعرض نص فى التقسيمات الفرعية للـ Panel بتعيين strings

لخاصية ال Caption بالتبادل ، ادخل ال Label objects فى ال Panel فرعية وعين نصاً لل Caption الخاصة بال Label- يمكنك أيضاً عرض نص معتم فى ال status panels بتحديد خاصية ال Enabled لل Label objects ب False .

• استخدم ال Format function ب Delphi لإنشاء formatted strings .  
تشبه هذه ال function ال sprintf() بال C وال C++ . تعد ال Format مفيدة فى إنشاء strings للعرض فى status panels ، ولكنها و function نافعة ايضاً لكثير من الاغراض .

• تعرض ال status panels الكلاسيكية مفاتيح ال Caps Lock ، ال Num Lock ، وال Scroll Lock ، وال Ins ، كما يوضح برنامج ال Caps Lock الخاص بهذا الباب ، استدع ال Windows GetKeyState function للحصول على المواصفات المرتبطة بالمفاتيح .

• يمكن ل Coolbar من لوحة ال Win32 ان يحمل band من windowed control مثل ال Win32 ToolBars وقائمة اللائحة . يستطيع المستخدمون إعادة ترتيب ال Coolbar bands فى وقت التشغيل .

• يقدم ال StatusBar من لوحة ال Win32 طريقة بسيطة لإنشاء ال statusbar متعدد ال bands فى النافذة .

فى الباب التالى ، ستعرف كيفية استخدام ال Delphis list component مثل ال ListBox وال ComboBox . سوف تفحص ايضاً ال list classes مثل ال TList ، ال TStrings ، وال TStringList .

# الباب الثامن

## تكوين الـ Lists

محتويات هذا الباب:

• Components

• List components

• String and other lists

ان كان هناك هيكل بيانات واحد يستخدمه المبرمج الآن أو فيما بعد، فكل مبرمج سوف يستخدم القائمة. فليس من المثير للدهشة ان واحداً من اهم الـ Windows controls هو list box. ومع هذا الـ control، والـ control المرتبط به وهو الـ combo box، والذي يضيف لـ text-entry area لـ list box، يمكنك إنشاء string list لأية اغراض اخرى. يمكنك ترتيب بيانات الـ list box ويمكنك الاختيار من بين انواع انماط مختلفة للـ list box والـ combo box - فمثلاً، ان تسمح بالاختيار المتعدد،

وكذلك يقدم Delphi عدة classes لصنع lists متعددة متعلقة ببعضها، مثل الـ TList، الـ TStringList، الـ TStrings. هذه هي كل الاسلحة النافعة في معركة إنشاء واجهة تطبيق معقدة للمستخدم، في معركة إنشاء واجهة تطبيق معقدة. للمستخدم، وكما ستعرف في هذا الباب، ان Delphi يقدم مجموعة غنية من الادوات للحفاظ على أى نوع من الـ list.

### :Components

فيما يلي قائمة الـ Delphi's components لصنع الـ list:



• **ComboBox**: يجمع ال Windows control المعيارى هذا بين ال ListBox و ال Edit object. وإعتماداً على ال ComboBox object ، يستطيع المستخدمون الاختيار من بين المداخل المذكورة فى ال list ، والتي تظهر اختياريًا فى drop-down ، أو يمكنهم إدخال بيانات جديدة فى ال Edit control . ال Standard : Palette .

• **ListBox**: استخدم ال Windows control المعيارى هذا لإنشاء strings lists والتي يمكن للمستخدمين ان يختاروا منها باستخدام اوامر الفأرة ولوحة المفاتيح . كما سيوضح هذا الباب ، تقدم ال ListBox فى TStrings array يسمى Items والذي يوفر وصولاً أسهل لبيانات ال ListBox . ال Standard : Palette .

• **StringGrid**: بالرغم من اسمها ، فإن هذا ال component قادراً على تخزين بيانات أو bitmap أو غيرها . وهذا يجعل ال StringGrid نافعة بشكل خاص فى إنشاء lists لل objects ذات الاسماء ، كما هو موضح فى تطبيق هذا الباب ال GlyphLst ، والذي يعرض اسماء ملفات bitmap لكل صورة ايقونة ملونة glyph يقدمها Delphi . ال Additional : Palette .

## : List Components

ان ال Delphi components المعيارين ، وهما ال ListBox وال ComboBox ، يمكن ان يتوليان أمر متطلبات اختيار ال list الخاصة ببرنامجك كله . كما ستعرف فى هذا الفصل ، ان Delphi يدعم ال controls ليوفر طريقة للوصول إلى البيانات الواردة فى ال list . على سبيل المثال ، بعبارات بسيطة ، يمكنك نقل ال ListBox strings وال ComboBox من والى ملفات النص . والتطبيق الخاص بهذا الفصل - وهو ال ToDo List utility - يوضح هذه التقنية ، وكذلك يوضح كيفية نقل strings بين اثنين من ال ListBox باستخدام واحدة من dialog-box templates الخاص بـ Delphi . يمكنك ان تجد تطبيق ال ToDo فى دليل ال Source\ToDo على القرص المدمج .

## : ListBoxes

يقدم ال ListBox عدداً من الخصائص الهامة التى يمكنك من الاختيار من بين اغمات مختلفة . على سبيل المثال ، عادة ما تكون ال ListBox لديها حد

## الباب الثامن : تكوين الـ Lists

خارجي ، ولكن يمكنك تحديد الـ `BorderStyle` بـ `bsNone` لتحصل على قائمة بلا حدود . (هذا التشكيل يبدو جيداً مع الـ `ListBox` المدخلة في `Bevel`) . عين قيمة الخاصية الـ `Columns` لعرض عناصر في `list` في أكثر من عمود . حسب النظام الافتراضي ، هذه القيمة محددة بصفر - حدها باثنين أو أكثر لإنشاء قوائم ذات أعمدة .

هناك طرق مختلفة عديدة لإدخال بيانات في الـ `ListBox` . في وقت التصميم ، اضغط الزر البيضاوي الخاصية الـ `Items` لفتح الـ `String list editor` الخاص بـ `Delphi` وادخل بنود قائمتك . أو ، يمكنك نسخ نص من ملف آخر في الـ `editor` هذا الـ `method` يخزن الـ `strings` المذكورة في قائمة في ملف `exe` . ويدخل تلقائياً الـ `strings` في الـ `ListBox` .

لإضافة `strings` في الـ `ListBox` في وقت التشغيل ، يمكن ان يستدعي البرنامج `methods` الخاصية الـ `Items` من الـ `TStrings class` كل `ListBox` يقدم `Items object` للتمكن من توصل اسهل الى بيانات الـ `list` . على سبيل المثال ، لحث المستخدمين على إضافة `strings` جديدة في الـ `ListBox` ، يمكنك استخدام البرمجة التالية مثل :

```
var
  S: String;
begin
  S := InputBox('Test Program', 'Enter a string', '');
  if Length(S) > 0 then
    ListBox1.Items.Add(S);
end;
```

لاحظ انك تستدعي الـ `Add method` للـ `Items object` الذي يملكه الـ `ListBox1` . ان الـ `ListBox` نفسه ليس له `Add method` - ، هذا الـ `method` ينتمي للـ `Items` . ولكن ، يمكنك تبسيط الـ `code` السابق باستخدام عبارة `with` مثل هذه :

```
with ListBox1.Items do
  Add(S);
```

ان عبارة with تخبر ال Delphi ان يستخدم method والتعريفات الاخرى في ال Items حسب النظام الافتراضى للحصول على strings out لل ListBox ، استخدم ال Items ك strings . على سبيل المثال ، حدد متغير ك strings مثل :

var

S: String;

بعد ذلك ، عين أى string لل ListBox S بتحديد ال index integer لل Item مع العبارة ، وعملية الوصول إلى ال Items فى ال array of Strings :  
S := ListBox1.Items[0];

ان المظاهر خداعة فى هذه الحالة لان ال Items ليس array فى الحقيقة - إنها خاصية ال TListBox class التى تستخدمها بنفس طريقة ال Pascal arrays . لتخزين بياناتها ال Items تملك خاصية Strings array مستخدماً فى عبارات تحدد قيم ال index فيما بين قوسين . والعبارة التالية ، على سبيل المثال ، مساوية تماماً للعبارة السابقة :

S := ListBox1.Items.Strings[0];

لان ال Strings array هو الافتراضى لل Items ، فإن الأسماء لا تحتاج الى تحديد . وفعل هذا لا يضر ، لذلك استخدم أى نوع من العبارة التى تفضلها . أى كان ال method الذى تستخدمه لفهرسة ال ListBox's strings ، إقصر الفهرس دائماً على المدى من (0) الى 1 - Items.Count . اذالم تفعل ، ال Delphi يحدث exception

استخدم for loop ، على سبيل المثال ، للتوصل لكل ال ListBox's strings :

for I := 0 to ListBox1.Items.Count - 1 do

begin

S := ListBox1.Items[I];

ShowMessage(S); { Or do something else with S }

end;

بالنسبة لل lists الطويلة ، قد تريد ان توفر بيانات من ملف نص يمكن للمستخدمين اعداده وتحريره مع ال Windows Notepad . لتحميل ملف النص فى ال ListBox ، استدع ال LoadFromFile method لل Items كما فى هذا المثال :

## الباب الثامن : تكوين ال Lists

```
ListBox1.Items.LoadFromFile('C:\data\file.txt');
```

استدع ال AddStrings لإضافة strings متعددة من احد ال ListBox (أو أى component آخر ذا خاصية ال TString) الى آخر . على سبيل المثال ، هذه العبارة تلحق ال strings من ال ListBox2 الى أية strings موجودة بالفعل فى ال ListBox1 :

```
ListBox1.Items.AddStrings(ListBox2.Items);
```

لتبادل بندى ال list ، استدع Exchange ، كما فى هذه ال code ، والذي يبذل ال strings الأول و الاخير فى ال ListBox1 :

```
with ListBox1.Items do  
  if Count >= 2 then Exchange(0, Count - 1);
```

لاحظ ، مرة اخرى هنا ، ان مدى ال strings هو من قيم ال index من (0) الى 1 - Items.Count لحذف strings من ال ListBox ، استدع ال Delete method ، والذي يتطلب items integer index . على سبيل المثال ، تحذف هذه العبارة البند الرابع من ال ListBox1 (ان فهرس البند الأول هو صفر ، لذا فإن الرابع يتم تعريفه بقيمة فهرس تساوى ٣) :

```
ListBox1.Items.Delete(3);
```

بدلاً من تخصيص فهرس حرفى ، أو فى الحالات التى لا تعرف فيها فهرس البند ، لكى تجد قيمة الفهرس ، استدع IndexOf ، والذي يدخل 1- اذا كان البند المحدد لا يوجد :

```
Index := ListBox1.Items.IndexOf('Item to find');
```

```
if Index >= 0 then  
  begin  
    { ... ok to use Index }  
  end;
```

استدع ال Clear method لحذف كل البنود من القائمة :

```
ListBox1.Items.Clear;
```

## Scrolling in listBoxes

تأتى ال ListBoxes بمفاجأة- اذا كان هناك سطور للعرض اكثر من المتاح داخل مساحة ال ListBox، يظهر scroll bar رأسى تلقائياً. عندما يستطيع ال ListBox عرض جميع البنود، يختفى scroll bar الرأسى تلقائياً.

تؤثر خاصيتان على ال scroll bar الرأسى فتشير ال ItemHeight الى ارتفاع لسطر النص الواحد بال pixels. حدد ال ItemHeight ب True لضمان ان سطور النص الكامل فقط تظهر فى حدود ال ListBox. حدد ال ItemHeight ب False لتسمح بسطور جزئية فى اسفل ال ListBox.

عندما تختار ال font لل ListBox، أو ل form تحتوى على ListBox، يقوم Delphi تلقائياً بتعديل ال ItemHeight الى قيمة مقابلة. للتغلب على هذا التعديل التلقائى، يجب ان تحدد ال Style ب lbOwnerDrawFixed أو lbOwnerDrawVariable. لمزيد من المعلومات عن انشاء owner-draw controls، انظر "استخدام ال TStrings class" لاحقاً فى هذا الباب.

يمكن أيضاً ان تعرض ال ListBoxes scroll bar افقى. لتشغيل ال scroll bar الافقى، استدع ال Perform method التابعة لل ListBox، والذي يرسل رسالة الى ال Windows ListBox الذى يقدم له ال ListBox component واجهة تطبيق مختصة بال objects. والرسالة التى يتم ارسالها فى هذه الحالة هى LB\_SETHORIZONTALEXTENT. ومع الرسالة، قم بتمرير مدى تحريك افقى بال pixels، زائد قيمة متغيرة ثالثة لم تستخدم والتى تكون دائماً صفر. على سبيل المثال، هذه العبارة تضيف scroll bar افقى لل ListBox، باقصى مدى تحريك وهو ١٠٠٠ pixels:

```
Listbox1.Perform(LB_SETHORIZONTALEXTENT, 1000, 0);
```

بدلاً من تمرير مدى حرفى مثل ال ١٠٠٠، فإن برنامجاً أفضل قد يحدد مدى التحريك الافقى بعرض اطول ال strings، بال pixels. توضح القائمة (٨-١) كيفية تحديد هذه القيمة. يستدعى البرنامج ال TextWidth فى ال Canvas الخاص بال ListBox (object يساعدك للوصول الى واجهة التطبيق الجرافيكية أو ال GDI). يدخل ال TextWidth عرض ال string، بال pixels، لل font

## الباب الثامن : تكوين الـ Lists

الحالي للـ Canvas . بعد تنفيذ الـ for loop ، يساوى متغير الـ K اقصى عرض الـ string ، بالـ pixels ، والذي يمررة البرنامج للـ ListBox مستخدماً رسالة الـ LB\_SETHORIZONTALEXTENT : Windows

**على قرص المدمج:** لكي نجعل مدى الـ scroll bar الافقى متماشياً مع بيانات الـ ListBox ، قم باداء الخطوات الموضحة فى القائمة (٨-١) بعد كل إضافة أو حذف ، بعد تغيير الـ font الـ ListBox . على القرص المدمج ، توجد هذه القائمة فى دليل الـ Source\Lists فى ملف Adjust.pas . استخدم هذه القائمة لتعديل مدى الـ scroll bar الافقى الخاص بالـ ListBox الى عرض اطول الـ list string ، بالـ pixels .



القائمة (٨-١): Code لتعديل مدى الـ scroll bar افقى للـ ListBox

Tvar

I, J, K: Integer;

begin

with ListBox1 do

begin

K := 0;

for I := 0 to Items.Count \_ 1 do

begin

J := Canvas.TextWidth(Items[I]);

if J > K then K := J;

end;

Perform(LB\_SETHORIZONTALEXTENT, K, 0);

end;

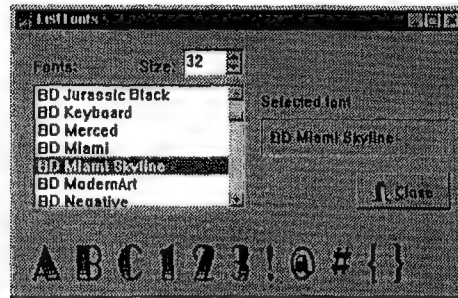
end;

ان الـ scroll bar افقى لا يوصى بها للـ ListBox متعددة الاعمدة . ولكن ، قد تعدل عرض العمود بارسال رسالة الـ LB\_SETCOLUMNWIDTH للـ ListBox باستخدام عبارة هذه :

ListBox1.Perform(LB\_SETCOLUMNWIDTH, 100, 0);

## إنشاء الـ ListBox:

إن تطبيق الـ ListFont الخاص بهذا الكتاب على القرص المدمج فى دليل الـ Source\ListFont يوضح كيفية إدخال قائمة بأسماء الـ font النظام فى الـ ListBox ، ثم استخدام تلك الأسماء لعرض string فى كل أسلوب الـ font. يوضح شكل (٨-١) عرض البرنامج. قم بتشغيل الـ ListFont واختر أى font لعرض النص بالنمط الافتراضى للـ font. اضغط زر الـ size لاختيار حجم نقطة جديد. تعتبر الـ TrueType Fonts فقط هى التى يمكن قياس كل أحجامها. توضح القائمة (٨-٢) الـ source code للبرنامج.



شكل (٨-١): يحمل الـ ListFont أسماء الـ Font النظام فى الـ ListBox

لإضافة list بأسماء البنط فى الـ ListBox، ينفذ برنامج الـ FontList العبارة التالية فى الـ OnCreate الخاص بالـ form:

```
FontListBox.Items := Screen.Fonts;
```

لتصنيف قائمة الـ font أبجدياً، حدد خاصية الـ Sorted للـ ListBox بـ True.

القائمة (٨-٢): Listfont\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
SysUtils, Windows, Messages, Classes, Graphics,
```

type

```
TMainForm = class(TForm)
    FontListBox: TListBox;
    Label1: TLabel;
    SampleLabel: TLabel;
    SpinEdit1: TSpinEdit;
    Label3: TLabel;
    FontNameLabel: TLabel;
    Label2: TLabel;
    Bevel1: TBevel;
    CloseBitBtn: TBitBtn;
    procedure FormCreate(Sender: TObject);
    procedure FontListBoxDblClick(Sender: TObject);
    procedure FontListBoxKeyDown(Sender: TObject;
        var Key: Word; Shift: TShiftState);
    procedure SpinEdit1Change(Sender: TObject);
    procedure CloseBitBtnClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    MainForm: TMainForm;
implementation
    {$R *.DFM}
    procedure TMainForm.FormCreate(Sender: TObject);
    begin
        FontListBox.Items := Screen.Fonts;
    end;
```



```

procedure TMainForm.FontListBoxDbClick(Sender:
TObject);
begin
  with FontListBox do
    if ItemIndex >= 0 then
      begin
        SampleLabel.Font.Name := FontListBox.Items[ItemIndex];
        FontNameLabel.Caption := SampleLabel.Font.Name;
      end;
end;

```

```

procedure TMainForm.FontListBoxKeyDown(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
  if Key in [VK_RETURN, VK_SPACE] then
    FontListBoxDbClick(Sender);
end;

```

```

procedure TMainForm.SpinEdit1Change(Sender: TObject);
begin
  SampleLabel.Font.Size := SpinEdit1.Value;
end;

```

```

procedure TMainForm.CloseBitBtnClick(Sender: TObject);
begin
  Close;
end;

```

end.

تستخدم الـ FontList تقنيتين لإختيار اسماء الـ font من الـ FontListBox. فى OnDbClick، تقوم عبارة if أولاً بفحص ما اذا كانت خاصية الـ ItemIndex للـ ListBox أكبر من أو تساوى صفر. اذا كانت كذلك، فيكون المستخدم قد اختار بنداً من الـ ListBox ولم يضغط مرتين الهواء الساخن.

## الباب الثامن : تكوين الـ Lists

وتقوم العبارتان المتتاليتان بتعيين اسم الـ font المختار من الـ Items لخاصية الـ Font.Name التابعة للـ SampleLabel. عند تعيين Name جديد لخاصية الـ font يتغير الـ Font للنص المعروض في اسفل النافذة. وأخيراً، يحدد الـ FontList اسم الـ font للـ Caption الخاص بـ label أخرى، والتي تعرض الـ font المختار.

في غالبية الحالات، انك تريد ان تمكن المستخدمين من اختيار بنود الـ ListBox بضغط الـ Enter وقضيب المسافة. يقوم برنامج العرض الـ ListFont بتشغيل هذه المفاتيح في الـ OnKeyDown الخاص بالـ FontList. يفحص الـ procedure ما اذا كان الـ Key موجود في المجموعة ذات القيمتين VK\_RETURN و VK\_SPACE، اذا كان كذلك، يستدع الـ FontListBoxDbClick. هذا عندما تضغط الـ Enter أو قضيب المسافة.

ان استخدام عامل in الخاص بالـ Pascal ومجموعة من القيم يعتبر اكثر فعالية، من تعبيرات الـ OR المنطقية المتعددة. على سبيل المثال، لاحظ الآتي:

```
if (Key = VK_RETURN) or  
    (Key = VK_SPACE ) or  
    (Key = VK_F9      ) then { ... }
```

بدلاً من العبارات السابقة، يمكنك اختبار ما اذا كان الـ Key في مجموعة من ثلاث قيم بهذه العبارة:

```
if Key in [VK_RETURN, VK_SPACE, VK_F9] then { ... }
```

### اختيار بيانات list:

في بعض الـ ListBoxes، تريد ان تمكن المستخدمين من ان يختاروا عنصر واحد فقط في المرة. في الـ ListBoxes الأخرى، تريد ان تمكنهم من اختيارات متعددة. ان اختيار عنصر واحد امر سهل - عليك فقط ان تستخدم مواصفات الـ ListBox الافتراضية مع خاصية الـ MultiSelect المحددة بـ False. لتشغيل الاختيار متعدد البنود، حدد الـ MultiSelect بـ True.

بالإضافة للـ MultiSelect، حدد الـ ExtendedSelect بـ True أو False، يكون الـ ExtendedSelect ليس له تأثير (يمكنك ان تتركه محدداً بقيمة الـ

True (الاصليّة) ولكن عندما يكون الـ MultiSelect محدد بـ True، الـ ExtendedSelect يحدد اختياراً متعدد البنود بواحدة من طريقتين:

• **ExtendedSelect = True**: يجب ان يضغط المستخدمون Ctrl ويضغطوا الفأرة لاختيار عناصر متعددة.

بالإضافة الى ذلك، فإن ضغط Shift يختار كل العناصر الواقعة بين السطر الذى تم اختياره سابقاً والسطر الذى تم ضغطه. ان ضغط Ctrl+Shift وضغط الفأرة يفعل نفس الشئ ويحجز ايه بنود اخرى يتم اختيارها بشكل فردى، وضغط عنصر مختار يؤدي الى إبطال اختياره.

• **ExtendedSelect = False**: لا يجب على المستخدمين ان يضغطوا ايه مفاتيح لاختيار بنود متعددة، ان اختيار عنصر جديد لا يزيد الإبراز عن الاختيارات السابقة.

يمكنك تعيين قيم الخصائص هذه فى نافذة الـ Object Inspector أو فى وقت التشغيل بعبارات مثل هذه:

```
ListBox1.MultiSelect := True;
```

```
ListBox1.ExtendedSelect := False;
```

عند تحديد الـ MultiSelect بـ True < والـ ExtendedSelect بـ False، تصبح مسئوليتك ان تزيل الاختيارات عن البنود المختارة- عندما يضغط المستخدمون Esc، مثلاً. والطريقة البسيطة لفعل هذا هى بربط خاصية الـ ExtendedSelect فتحاً وإغلاقاً كما توضح القائمة (٨-٣) استخدم هذا الـ code فى OnKeyDown الخاص بالـ ListBox.

عند تحديد الـ MultiSelect بـ True، تشير خاصية الـ SelCount الى عدد العناصر المختارة. اذا لم يكن هناك بنود مختارة، يساوى الـ SelCount صفراً. عند تحديد الـ ExtendedSelect بـ False، استخدم OnKeyDown هذا لإبطال إبراز البنود المختارة فى ListBox عندما يضغط المستخدمون Esc.

القائمة (٨-٣): OnKeyDown لإبطال إبراز العناصر المختارة

```
procedure TForm1.ListBox1KeyDown(Sender: TObject;
```

## الباب الثامن : تكوين الـ Lists

```
var Key: Word; Shift: TShiftState);
begin
  if Key = VK_ESCAPE then
    with ListBox1 do
      begin
        ExtendedSelect := True;
        ExtendedSelect := False;
      end;
    end;
end;
```

ان الـ SelCount دائماً يساوى 1- اذا كان الـ ExtendedSelect محدد بـ False ، بغض النظر عما اذا كان البند مختاراً . عندما يكون الـ MultiSelect محدد بـ True ، يشير الـ ItemIndex الى احدث عنصر تم ضغطه بغض النظر عما اذا كان هذا العنصر مختاراً أو غير مختار . وهذه الحالات الشاذة قد تحيرك . للحصول على افضل النتائج ، لا تستخدم الـ SelCount مع الـ ListBoxes احادية المداخل ، ولا تستخدم الـ ItemIndex مع الـ ListBoxes متعددة المداخل .

تقترح القائمة (٨-٤) احدى طرق الحصول على اختيارات متعددة من الـ ListBox . يجب ان يكون الـ MultiSelect محدد بـ True حتى تعمل هذه الـ code بشكل مناسب . لتجرب هذه التقنية ، اتبع هذه الخطوات :

- ١- أضف أثنان من الـ ListBox على الـ form .
- ٢- أضف بعض الـ strings فى خاصية الـ Items للـ ListBox1 . حدد خاصية الـ MultiSelect للـ ListBox1 بـ True .
- ٣- أضف Button على الـ form واستخدام الـ list فى إنشاء الـ OnClick الخاص بالـ Button .
- ٤- قم بتشغيل البرنامج واضغط الزر لنقل البنود المختارة من الـ ListBox1 الى الـ ListBox2 .

### اختيار عناصر الـ ListBox فى وقت التشغيل،

فى بعض الاحيان يكون اختيار عناصر الـ ListBox باستخدام عبارات البرنامج أمراً نافعاً- على سبيل المثال ، لاستعادة نافذة برنامج الى حالة الحفظ أو

لعمل اختيارات سابقة التحديد من مجموعات بيانات . ان ضغط الزر A قد يؤدي الى اختيار بنود 3، 6، و 9 تلقائياً؛ وضغط الزر B يختار بنود 2، و 7؛ الى آخره .

القائمة (٨-٤): الحصول على اختيارات متعددة من الـ list box

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  ListBox2.Clear;
  for I := 0 to ListBox1.Items.Count - 1 do
    if ListBox1.Selected[I] then
      ListBox2.Items.Add(ListBox1.Items[I]);
end;
```

لإختيار بنود فى ListBox أحادى المدخل فى وقت التشغيل ، عين قيمة لخاصية الـ ItemIndex . على سبيل المثال ، تختار العبارة التالية البند السادس (إن فهرس البند الأول هو صفر ، لذا فإن فهرس البند السادس يساوى خمسة) :

```
Listbox1.ItemIndex := 5;
```

ولكن هذا لا يعمل عندما تكون الـ MultiSelect محددة بـ True . فى هذه الحالة ، لإختيار عناصر فى وقت التشغيل ، عين True لمدخلات الـ Selected array . فمثلاً ، تختار العبارتان التاليتان البند الثالث والسادس فى الـ ListBox1 . يقوم Delphi تلقائياً بتعديل الـ SelCount والـ ItemIndex للقيم السليمة التالية لكل تعيين :

```
Listbox1.Selected[2] := True;
```

```
Listbox1.Selected[5] := True;
```

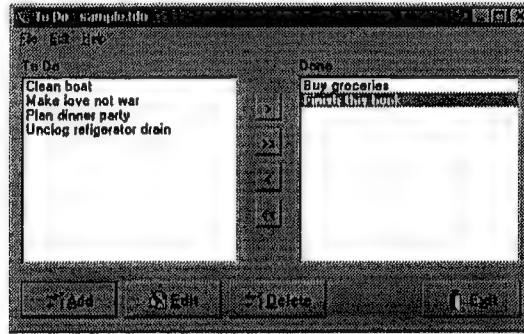
لأن Selected array لا يمكن التوصل إليه عندما تكون الـ MultiSelect محددة بـ False ، فقد تريد استخدام عبارة if مثل التالية لإختيار البند السادس فى الـ ListBox1 . يعمل هذا الـ code بشكل سليم للـ ListBoxes ذات المداخلات المتعددة والأحادية .

## الباب الثامن : تكوين الـ Lists

```
with ListBox1 do
if MultiSelect then
    Selected[5] := True
else
    ItemIndex := 5;
```

### استخدام بيانات الـ ListBox:

كمثال عملي على استخدام كلا الـ ListBoxes البيانات فيما بينهما باستخدام الـ Button objects، يعرض تطبيق الـ ToDo قائمة بسيطة للمشروعات التي يمكنك عملها. لقد استخدمت template من Dual list box الخاصة بـ Delphi لإنشاء نافذة البرنامج. ثم قمت بتعديل stock form لبرنامج الـ ToDo. يوضح شكل (٨-٢) عرض الـ ToDo. وخلال باقى هذا الباب، سأوضح بعض code الـ ToDo، والتي يصعب ذكرها هنا كاملة بسبب طولها.



شكل (٨-٢): يوضح تطبيق الـ ToDo كيفية استخدام template من Dual list box الخاص بـ Delphi لنقل البيانات بين الـ ListBox objects

تقدم template من Dual list box الـ code اللازم لنقل البيانات بين اثنين من الـ ListBox objects. لفحص هذا الـ code أضف from جديدة فى مشروع واختر مربع قائمة الـ Dual من template gallery. اعرض الـ form unit لترى كيف يتحكم Delphi فى كلا من الـ ListBoxes.

### إنشاء Sorted ListBoxes:

كما ذكرت، إن إنشاء ListBoxes المرتبة يعد أمراً سهلاً- فقط حدد خاصية الـ Sorted بـ True. يحافظ الـ ListBox تلقائياً على ترتيب معين للإضافات

والمحذوفات. ولكن لأن الـ Windows هو الذى يؤدى الترتيب، وليس الـ Delphi component. على سبيل المثال، بتعيين "Zebra" للبند الأول للـ ListBox1:

```
ListBox1.Items[0] := 'Zebra';
```

لوضع الـ Zebra فى نهاية الحظيرة حيث ينتمى غالباً، اجعل خاصية الـ Sorted مساوية الى on. هذه العبارات تجبر الـ ListBox أن يستعيد بياناته:

```
ListBox1.Sorted := False;
ListBox1.Sorted := True;
```

### ١٠ الـ ComboBoxes

إن الـ ComboBoxes رائعة فى إنشاء قوائم إختيار ذات إمكانات عرض إختيارية. إن الـ ComboBox يدمج بين Edit Box (يشبه Edit الخاص بـ Delphi) مع الـ ListBox. يستطيع المستخدمون إختيار بنود من القائمة ويمكنهم إدخال بيانات جديدة فى نافذة الـ edit control. وأنماط الـ ComboBox الأساسية الثلاثة هى:

● **Simple (Style = csSimple):** يكون الـ ListBox دائماً مرئياً. يستطيع إدخال بند جديد فى نافذة الـ edit.

● **Drop-Down (Style = csDropDown):** يسقط الـ ListBox عندما المستخدمون زر سهم الإسقاط المجاور لنافذة الـ edit - يمكنهم أيضاً ضغط **Alt+Down**. كما فى نمط الـ select، يستطيع المستخدمون إختيار بند من القائمة، أو يمكنهم إدخال عنصر جديد فى الـ edit. إن البحث المتزايد أيضاً قد تم تشغيله لهذا النمط - اكتب رمزاً واحداً أو أكثر واضغط مفتاح السهم العلوى أو السفلى لإختيار المدخل الذى يتماشى بشكل وثيق مع مدخلاتك.

● **Drop-Down-List (Style = csDropDownList):** كما هو الحال فى نمط الـ Drop-Down، يسقط الـ ListBox عند ضغط زر سهم الإسقاط. مع هذا الأسلوب الـ edit box سيكون "read-only"، ويجب أن يختيار المستخدمون مدخل مذكور فى القائمة. ولكن، قد يكتب المستخدمون الحرف

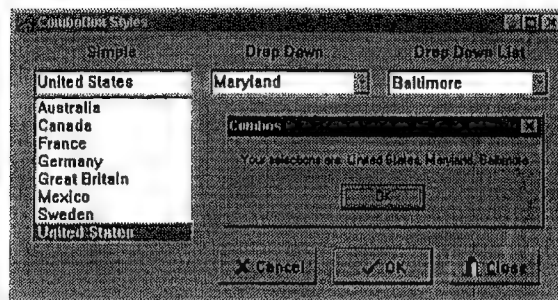
## الباب الثامن : تكوين الـ Lists

الأول من البند لإختياره . على سبيل المثال ، اضغط مفتاح الـ P متكرراً لإختيار كل المداخل التي تبدأ بالـ P .

**ملحوظة:** على عكس الـ ListBox ، لا تستطيع الـ ComboBoxes عرض scroll bar أفقي . هذا الحد خاص بالـ Windows .



يوضح تطبيق الـ Combos أساليب الـ ComboBox الثلاثة . يوضح شكل (٣-٨) عرض البرنامج . توضح القائمة (٨-٥) الـ source file .



شكل (٣-٨) : يوضح تطبيق الـ Combos أنماط الـ ComboBox الثلاث : Simple، Drop-Down، و Drop-Down-List

توضح قائمة الـ Combos الطريقة الصحيحة للحصول على إختيارات من الـ ComboBox . في غالبية الحالات ، يجب أن تفعل هذا بالإشارة الى خاصية الـ Text للـ object والتي تحمل النص المعروض في نافذة edit control . قد تستخدم الـ ListBox method للحصول على إختيارات ، مثلاً ، بالإشارة الى الـ Items : array

```
if ItemIndex >= 0 then
  S1 := SimpleCB.Items[ItemIndex]; // ???
```

القائمة (٨-٥) : Combos\Main.pas

```
unit Main;
interface
uses
  SysUtils, Windows, Messages, Classes, Graphics,
```



---

Controls, Forms, Dialogs, StdCtrls, Buttons;

type

```

TMainForm = class(TForm)
    SimpleCB: TComboBox;
    DropDownCB: TComboBox;
    DropDownListCB: TComboBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    CancelBitBtn: TBitBtn;
    OKBitBtn: TBitBtn;
    CloseBitBtn: TBitBtn;
    procedure CancelBitBtnClick(Sender: TObject);
    procedure OKBitBtnClick(Sender: TObject);
    procedure CloseBitBtnClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

```

var

```

MainForm: TMainForm;
implementation
{$R *.DFM}

```

```

procedure TMainForm.CancelBitBtnClick(Sender: TObject);
begin
    Close;
end;

```

```

procedure TMainForm.OKBitBtnClick(Sender: TObject);

```

## الباب الثامن : تكوين الـ Lists

```
var
    S1, S2, S3: string;
begin
    S1 := SimpleCB.Text;
    S2 := DropDownCB.Text;
    S3 := DropDownListCB.Text;
    ShowMessage('Your selections are: ' +
        S1 + ', ' + S2 + ', ' + S3);
end;

procedure TMainForm.CloseBitBtnClick(Sender: TObject);
begin
    OKBitBtn.Click; { Simulate OK button click }
    Close;          { End program }
end;

end.
```

وهذه التقنية موضع استفسار لأنها لا تتفق بشكل سليم مداخل المستخدم من حقل الـ Text . للحصول على أفضل النتائج ، احضر الاختيارات من خاصية الـ Text إلا اذا كنت تحتاج الى أداء أعمال على المداخل فى نافذة الـ ListBox . على سبيل المثال ، تنسخ هذه العبارة الاختيار من الـ SimpleCB ComboBox الى متغير S1 :

```
S1 := SimpleCB.Text;
```

فى وقت التصميم ، أدخل بنود قائمة فى خاصية الـ Items للـ ComboBox . أدخل أو امسح خاصية الـ Text لتغيير مساحة الـ string edit ، والى يعين لها Delphi اسم الـ object حسب النظام الافتراضى .

الـ ComboBox يقدم method أخرى قد تجدها نافعة فى تصميم واجهة تطبيق برنامجك . استخدم خاصية الـ DroppedDown لتحديد ما اذا كان الـ ComboBox معلق الى أسفل :

```
if DropDownCB.DroppedDown then
    { ... do something if ComboBox is open }
```

عين بـ True أو False للـ DroppedDown لفتح أو إغلاق جزئية القائمة للـ ComboBox الواقعة تحت البرنامج. على سبيل المثال، أضف ComboBox و Button على الـ form، واستخدم هذه العبارة onClick الخاص بالـ Button لفتح وإغلاق نافذة قائمة الـ ComboBox:

```
with ComboBox1 do
  DroppedDown := not DroppedDown;
```

لإختيار كل الـ text فى مساحة الـ ComboBox's edit، استدع الـ SelectAll method. هذا يبرز نص الـ edit- إنه لا يختار كل بنود القائمة كما يبدو من اسم الـ method. على سبيل المثال، يمكنك استخدام هذه العبارة فى OnKeyDown لإبراز نص الـ edit عندما يضغط المستخدمون مفتاحاً بعينه:

```
SimpleCB.SelectAll;
```

إن الـ ComboBoxes هى objects أحادية المدخل. إن الـ ListBox components فقط هى التى تتعامل مع الإختيارات المتعددة.

إن استدعاء الـ Clear يسمح كل مداخل القائمة وأى نص فى edit box من الـ ComboBox object. لحذف كل النص من الـ ComboBox، استخدم هذه العبارة:

```
ComboBox1.Clear;
```

إذا أردت حذف النص فقط من الـ edit، عين الـ null string لخاصية الـ Text للـ ComboBox:

```
ComboBox1.Text := "";
```

يمكنك أيضاً استخدام الـ methods الموضحة سابقاً لعمل التغييرات ببند القائمة من خلال الـ Items array. فمثلاً، هذا يضيف مدخل جديد لمربع الـ ComboBox's list:

```
ComboBox1.Items.Add('Aardvark');
```

## :String and Other Lists

تعتبر الـ ListBox والـ ComboBox أدوات بصرية لإنشاء قوائم إختيار بالنوافذ. تحتاج غالبية البرامج أيضاً إلى إمكانات صنع قوائم داخلية باستخدام

## الباب الثامن : تكوين الـ Lists

objects ليست فى لوحة الـ VCL الخاصة بـ Delphi. يوضح هذا الفصل ثلاث classes يمكنك استخدامها للتحكم فى string list الـ list الأخرى فى تطبيقاتك. حسب الترتيب الأبجدي، الـ classes الثلاث هى :

### • TList class لصنع قائمة.

• **TStringList**: وهى class يمكن للتطبيقات استخدامها لإنشاء قوائم للـ string والـ objects. تحدد الـ TStringList الذاكرة اللازمة لتخزين الـ string. استخدم هذا النوع لإنشاء متغيرات مستقلة عندما تحتاج مكاناً لتخزين قوائم الـ Strings.

• **TStrings**: وهى abstract class تستخدمها خصائص الـ component للتحكم فى قوائم الـ strings الـ objects الأخرى. لا تقوم الـ TStrings بتخصيص أية ذاكرة للتخزين. إن المرة الوحيدة التى تستخدم هذا النوع من القوائم هى عندما تشير الى خصائص الـ component object. إنك لا تنشئ أبداً متغيرات مستقلة من هذا النوع.

**ملحوظة:** إن الـ classes السابقة ليست أدوات واجهة تطبيق بالمعنى المفهوم، ولكننى أريد أن أقدمها هنا فى الجزء الثانى من الكتاب بسبب أهميتها البالغة فى تطوير تطبيقات Delphi. ولكن، إننى أشرح أيضاً الـ StringGrid component، والذى يستخدم قوائم الـ string وهو أداة قيمة جداً لإنشاء واجهات تطبيق عملية للمستخدم.

### استخدام الـ TList class:

إن Delphi يستخدم الـ TList class كأساس لإنشاء قوائم ذات أغراض عامة. يمكنك أيضاً استخدام الـ TList كـ class لتخزين الـ objects من أى نوع فى قوائم. فى أغلب الحالات، لكى تستخدم الـ TList، إنك تحتاج لإنشاء إثنين من الـ class - واحدة لبنود الـ objects للتخزين فى القائمة، وواحدة للقائمة ذاتها. على سبيل المثال، يمكنك إنشاء class فى جزئية تنفيذ unit module. إن العناصر الموجودة فى القائمة هى objects من نوع الـ class الخاص بك، والتى يمكن تعريفها فى واجهة التطبيق للوحدة مثلما يلى :

```
type
  TAnyItem = class
    Data: Integer;
    constructor Create(Data1: Integer);
  end;
```

كما هو مقرر هنا، تعتبر الـ TAnyItem نوع بيانات class توفر متغير عدد صحيح يسمى Data. بالرغم من عدم الاستخدام العملي، تظهر الـ TAnyItem الاسس لإنشاء قائمة للـ objects. بالإضافة الى عضو الـ Data التابع لها، فإن الـ class لها constructor يسمى Create لبدء تكوين الـ objects بالـ class. اجعل دائماً اسم constructor هو Create. نفذ الـ constructor كما تفعل مع أى procedure، ولكن استخدام كلمة constructor الأساسية فى مكان الـ procedure، وإسبق اسم الـ method بـ TAnyItem ونقطة. ما يلى يوجد فى قطاع الـ implementation:

```
constructor TAnyItem.Create(Data1: Integer);
begin
  Data := Data1;
  inherited Create;
end;
```

يحفظ الـ constructor parameter الـ Data1 فى متغير الـ Data الخاصة بالـ object ثم يستدعى الـ object Create المشتقة من الـ ancestor class. إن استدعاء الـ Create المشتق يعطى الـ ancestor class الفرصة لأداء البدء الخاص بها، أى كانت. (فى البرمجة المختصة بالـ object، إننا لا نحتاج أن نعرف كل التفاصيل الدقيقة عن ancestor class methods). فى هذه الحالة، الـ ancestor class هى الـ TObject، الـ class الأم لكل Delphi classes، حتى تلك التى تنشئها أنت بنفسك.

**ملحوظة:** اعتبر الـ class constructor كمنشئ وللعناصر الداخلية لـ object جديد. لكل object من الـ class، يجب ان يستدعى البرنامج الـ Create constructor لبدء الـ object.

Note

## الباب الثامن : تكوين الـ Lists

إنك تحتاج أيضاً class للقائمة. لا يوجد سبب وجيه لإعادة ابتكار class، ان تستغل الـ methods التي توفرها الـ TList، ان تستخرج الـ class الجديدة من الـ TList. ترث الـ class الجديدة الخصائص والـ methods من الـ TList. بعبارة أخرى، ان الـ class الجديدة هي نفسها الـ TList، زائد أى إمكانيات تقرر إضافتها. لإنشاء class مشتقة، استخدم هذا التعريف قطاع فى الـ unit's implementation :

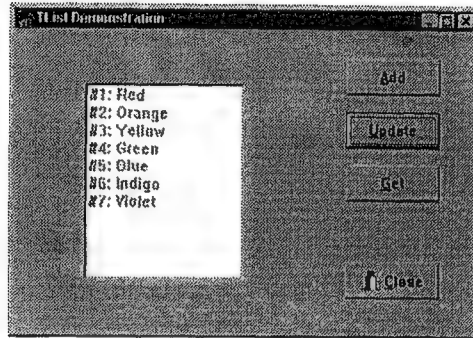
```
type
  TAnyList = class(TList)
    destructor Destroy; override;
  end;
```

الـ TAnyList class تستخدم الـ override للـ Destroy destructor فى الـ TList class ancestor. وهذا ضرورى لان الـ TList objects فى العادى لا تدمر الـ objects التى تحتوها. لإضافة هذه الامكانية للـ class المشتقة، قم بتنفيذ الـ destructor من خلال قطاع unit's implementation، كما يلى :

```
destructor TAnyList.Destroy;
var
  I: Integer;
begin
  for I := 0 to Count - 1 do
    TAnyItem(Items[I]).Free;
  inherited Destroy;
end;
```

أولاً، تستدعى الـ for loop الـ Free لكل الـ TAnyItem objects على القائمة. تحافظ الـ TList class على بنود الـ Count. تحتوى خاصية الـ Items على pointer تشير الى (عناوين) الـ objects فى القائمة. لإستدعاء method مثل الـ Free يتطلب تعبير type-cast كما هو موضح فى هذا المثال، والذي يخبر الـ compiler ان الـ Items[I] ليس مجرد pointer، ولكن الـ TAnyItem object. أخيراً، يستدعى الـ destructor الجديد الـ Destroy المشتق لإعطاء الـ TList class فرصة لأداء المسح الخاص به.

يمكنك، وأنت مسلح بهذه الأسس، أن تستخدم الـ TList لإنشاء قوائم من أى نوع من الـ objects. يظهر برنامج الـ StrList الخطوات اللازمة. يوضح شكل (٤-٨) عرض البرنامج. تعطى القائمة (٦-٨) الـ source code. قم بتشغيل البرنامج، اضغط Add، وإدخل string. أعد هذه الخطوات عدة مرات لإنشاء قائمة الـ string objects فى الذاكرة.



شكل (٤-٨): يوضح تطبيق الـ StrList كيفية استخدام  
الـ TList class بـ Delphi's لإنشاء قوائم من الـ objects فى الذاكرة

اضغط Update لنقل الـ objects القائمة الى الـ ListBox. (لن ترى الـ strings الخاص بك حتى بعد ان تضغط الـ Update). اضغط Get وإدخل strings للبحث فى القائمة. سوف اشرح المزيد عن الـ code بعد القائمة.

#### القائمة (٦-٨): Strlist\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
SysUtils, Windows, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, StdCtrls, Buttons;
```

```
type
```

```
TMainForm = class(TForm)
```

```
AddButton: TButton;
```

## الباب الثامن : تكوين ال Lists

////////////////////////////////////

```
    CloseBitBtn: TBitBtn;
    GetButton: TButton;
    ListBox1: TListBox;
    UpdateButton: TButton;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure AddButtonClick(Sender: TObject);
    procedure UpdateButtonClick(Sender: TObject);
    procedure GetButtonClick(Sender: TObject);
    private
    { Private declarations }
    public
    { Public declarations }
    end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

type

    {- New class of items to insert in a list }
    TStrItem = class
        S: string;
        I: Integer;
        constructor Create(S1: string; I1: Integer);
    end;

    {- Derived class to hold TStrItem objects }
    TStrList = class(TList)
        destructor Destroy; override;
```



```

function FindItem(S1: string): TStrItem;
end;

var
  StrList: TStrList;      { List of TStrItems }
  ItemCount: Integer;     { Number of items inserted }

{- Create a new instance of TStrItem }
constructor TStrItem.Create(S1: String; I1: Integer);
begin
  S := S1;                { Save string parameter }
  I := I1;                { Save integer parameter }
  inherited Create;       { Call inherited Create }
end;

{- Destroy instance of TStrList }
destructor TStrList.Destroy;
var
  I: Integer;
begin
  for I := 0 to Count - 1 do
    TStrItem(Items[I]).Free; { Free all TStrItems }
  inherited Destroy;        { Call inherited destroy }
end;

{- Return object identified by S1 or nil for no match }
function TStrList.FindItem(S1: string): TStrItem;
var
  I: Integer;
  P: TStrItem;
begin
  for I := 0 to Count - 1 do
    begin

```

## الباب الثامن : تكوين الـ Lists

```
P := TStrItem(Items[I]); { P refers to a TStrItem object }
  if Uppercase(P.S) = Uppercase(S1) then { Match? }
  begin      { Found match }
    Result := P; { Return P as function result }
    Exit;      { Exit function immediately }
  end;
end;
Result := nil; { No match; return nil }
end;
{- TMainForm event handlers }

procedure TMainForm.FormCreate(Sender: TObject);
begin
  StrList := TStrList.Create; { Create new StrList object }
  ItemCount := 0;             { Initialize insertion count }
end;
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  StrList.Free; { Also destroys listed items }
end;
{- Button event handlers }
procedure TMainForm.AddButtonClick(Sender: TObject);
var
  StrItem: TStrItem; { New item to insert }
  S: string;         { User input string }
begin
  S := "";
  if InputQuery(Caption, 'Enter item', S) then
    if Length(S) > 0 then
      begin
        Inc(ItemCount);
        StrItem := TStrItem.Create(S, ItemCount);
        StrList.Add(StrItem);
```

```

end;
end;

procedure TMainForm.UpdateButtonClick(Sender: TObject);
var
  I: Integer;
  P: TStrItem;
begin
  ListBox1.Clear;
  with StrList do
    for I := 0 to Count - 1 do
      begin
        P := TStrItem(StrList.Items[I]);
        ListBox1.Items.Add(Format('#%d: %s', [P.I, P.S]));
      end;
    end;
end;

procedure TMainForm.GetButtonClick(Sender: TObject);
var
  S: string;
  P: TStrItem;
begin
  UpdateButton.Click;
  S := '';
  if InputQuery(Caption, 'Enter item name', S) then
    if Length(S) > 0 then
      begin
        P := StrList.FindItem(S);
        if P <> nil then
          ShowMessage(Format('%s, Number = %d', [P.S,
            P.I]));
        else
          ShowMessage('No such item');
        end;
      end;
    end;
end;

```

end;

end.

يظهر الـ StrList كيفية إنشاء قائمة مرتبطة، والتي تتطلب إثنان من الـ classes. الـ class الأولى، TStrItem، تعرف إثنين من أعضاء البيانات- الـ string S، وقيمة العدد الصحيح I. تعرف الـ class أيضاً Create constructor لبدء هذه المتغيرات. والـ class الثانية، TStrList، والمتشقة من الـ TList، تعرف إثنان من الـ methods: destructor و function، FindItem، التي تبحث في القائمة عن object معرف بمتغير string.

لإنشاء object القائمة، يعرف البرنامج متغيراً، وهو StrList، من الـ TStrList class. إذا كنت تفضل، يمكن أن تذهب هذه المتغيرات الى الـ form class.

الـ TStrItem constructor يعين الـ parameter عدد صحيح و string لمتغيرات الـ I والـ S الخاصة بالـ class. هذا يبدأ في إدخال objects جديدة في القائمة. لاحظ أن الـ constructor يستدعي الـ Create الموروث، بالرغم من أن الـ TStrItem تعد class جديدة ليست مشتقة من class أخرى. وهذه الخطوة ليست ضرورية، ولكن يجب أن تفعلها على أية حال لأن، في Pascal الخاصة بـ Delphi، تعتبر كل الـ classes مشتقة من الـ class الـ TObject.

**فكرة:** لأن الـ classes الجديدة تعتبر مشتقة من الـ TObject بشكل تلقائي، لذا فإن كل Delphi classes و objects تعتبر مرتبطة ببعضها. من بين المميزات الأخرى، هذا يعنى أنه بإمكانك تمرير object من أى نوع class الى الـ TObject parameter فى الـ procedure أو الـ function.

توفر الـ TObject، Create constructor وإفترضى و Destroy destructor إفترضى، زائد code لإرسال رسالة. باختيار library source Delphi code وقت التشغيل، تجد تعريف Pascal الخاص بالـ TObject فى ملف الـ System.pas، الموجود فى المسار Source\Rtl\Sys وكما ستعرف بتصفح هذه المعلومة، إن الـ TObject methods مكتوبة باستخدام عبارات لغة الـ Assembly. ولقد جرت العادة على أن يتم تخزين هذه الـ code منفصلة فى

ملفات Clsh.asm و Clsf.asm؛ وهى الآن فى System.pas. (أنك لا تحتاج الـ source code لاستخدام الـ TObject، ولكن اذا قمت بكثير من البرمجة فى Delphi، فيمكنك معرفة الكثير بتصفح component methods مثل هذه). إنك لا تحتاج الى اشتقاق classes من الـ TObject. على سبيل المثال، التعريفان للـ class التاليين يعتبران متساويين:

```
type
  TNewClass1 = class
    {...}
  end;
  TNewClass2 = class(TObject)
    {...}
  end;
```

ان الـ TStrList destructor، يستدعى Free لكل الـ TStrItem object على القائمة. ثم يستدعى الـ Destroy الموروث. قد يستخدم code مشابهة فى method اخرى - مثلاً، واحداً يسمح الـ objects من القائمة. تبحث الـ Function FindItem فى قائمة الـ objects عن objects يتماشى مع string parameter. تتكرر الـ for loop فى القائمة باستخدام الـ Count الخاص بالـ TList. لتبسيط الـ code، تعين العبارة كل الـ objects فى القائمة الى الـ P:

```
P := TStrItem(Items[I]);
```

ان P هو متغير من (TStrItem) list object class، ولكن بالرغم من المظاهر، ان الـ P لا يحتوى على بيانات حالة الـ object. يحتوى الـ P على pointer reference الى الـ TStrItem object instance. فى Delphi، تعتبر كل متغيرات الـ class-object، مثل الـ P، reference أى انها pointers تشير الى الـ objects فى الذاكرة. اذا كنت تعرف نسخ اخرى من الـ Pascal، قد يبدو الـ code السابق على انها تنسخ TStrItem object بأكمله للـ P، ولكن هذا لا يحدث فى الـ Pascal Object بـ Delphi. ان كل ما يتم تعيينه هو العنوان التابع الـ TStrItem object فى الـ Items array بالقائمة. تعتبر كل متغيرات الـ class - أى الـ objects - تعتبر references.

## الباب الثامن : تكوين الـ Lists

داخل الـ for loop ، تقارن عبارة الـ if الـ string object (P.S) بالقيمة المتغيرة (S1) التي تم تمريرها الى الـ FindItem . تستخدم العبارة الـ Uppercase function بـ Delphi لمقارنة الـ strings بغض النظر عن الاحرف . اذا توافقت الـ strings ، يعين البرنامج الـ P للـ Result الخاصة بالوظيفة ، ثم يخرج الـ function على الفور باستدعاء الـ Exit procedure بالـ Pascal . اذا لم يتوافقا ، تعين العبارة النهائية لاشئ للـ Result الخاص بالـ function .

بالرغم من ان الـ function للـ FindItem هي الـ TStrItem ، تقوم الـ function في الواقع بادخال reference الى الـ TStrItem object . هذه هي فائدة اخرى لمساعدة Delphi ان كل متغيرات الـ class-object تعتبر references . وبالتالي ، يسمح Delphi بتعين لاشئ للـ Result الخاص بالـ FindItem ليشير الى انه لا يرجع الى أى object .

يقوم الـ OnCreate الخاص بالـ StrList للـ form بانشاء قائمة جديدة بتنفيذ العبارة التالية ، والتي تستدعي الـ Create method الخاص بالـ TStrList . وتقوم عبارة تعيين بانشاء object TStrList في الذاكرة ، وتبدأ متغير الـ TStrList ليعود على الـ object :

```
StrList := TStrList.Create;
```

ان احدى قواعد Delphi الاساسية هي انك يجب ان تمحو الـ objects التي تنشئها . في هذه الحالة الـ OnDestroy بتحرير object القائمة باستدعاء الـ Free method الموروث من الـ TList :

```
StrList.Free;
```

ولان الـ TStrList destructor يحو كل الـ objects القائمة بشكل سليم يتخلص من كل الذاكرة المخصصة للقائمة والـ objects التي تحتوى عليها .

لإضافة objects جديدة للقائمة ، فإن الـ OnClick الخاص بالـ AddButton's يحثك على إضافة string تنشئ هاتان العبارتان TStrItem object جديد وتضيفاه للقائمة باستدعاء الـ Add method للـ TList :

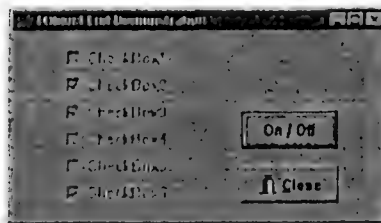
```
StrItem := TStrItem.Create(S, ItemCount);
```

```
StrList.Add(StrItem);
```

في البرنامج المثال، عندما تضغط زر ال Update، يقوم ال onClick الخاص به بنقل كل ال string والقيمة العددية الصحيحة الخاصة بال object الى ال ListBox component object. يرجع التعبير P.I و P.S على البيانات الموجودة في كل ال TStrItem object في القائمة. وتعرض تعبيرات مشابهة في ال GetButtonClick اعداد صحيحة و object strings بعد استدعاء ال FindItem للبحث في القائمة.

### إنشاء Array of TObject References

في التطبيقات المعقدة، من الصواب ان تنشئ array من ال TObject references لل components المتنوعة في ال form. ان كل ال classes تنحدر من ال TObject، ولذلك فإن ال TObject array يمكنها ان تعود على أى Delphi object. على سبيل المثال، يمكنك إدخال مجموعة من component objects في ال TObject array، ثم تستخدم ال array لاداء عمليات على كل ال objects. يوضح تطبيق ال ObjList [انظر القائمة (٧-٨)] التقنيات اللازمة. وتحتوى نافذة ال form على ستة من ال CheckBox objects. اضغط زر ال On/Off الخاص بال form لابطال وتشغيل ال CheckBoxes. ينفذ البرنامج هذه المهمة بتعريف ال pointer الى ال TObject array: ثم استخدم ال GetMem لتخصيص ذاكرة لعدد ال CheckBoxes بال form. يقوم ال OnCreate الخاص بال form بتعيين كل ال CheckBoxes الى عناصر ال CheckBoxArray (تذكر ان جميعها references). ويستخدم ال onClick الخاص بال array for loop لكي يجعل كل خاصية ال Enabled لـ CheckBox object بـ on أو off. أخيراً، يستدعى ال OnDestroy الخاص بال form ال FreeMem للتخلص من الذاكرة المخصصة. يوضح شكل (٥-٨) عرض البرنامج.



شكل (٥-٨): يوضح ال ObjList كيفية إنشاء array لل TObject references لاداء عمليات على مجموعات ال objects

unit Main;

interface

uses

SysUtils, WinTypes, WinProcs, Messages, Classes,  
Graphics,

Controls, Forms, Dialogs, StdCtrls, Buttons;

type

TMainForm = class(TForm)

  CheckBox1: TCheckBox;

  CheckBox2: TCheckBox;

  CheckBox3: TCheckBox;

  CheckBox4: TCheckBox;

  CheckBox5: TCheckBox;

  CheckBox6: TCheckBox;

  OnOffButton: TButton;

  BitBtn1: TBitBtn;

  procedure FormCreate(Sender: TObject);

  procedure FormDestroy(Sender: TObject);

  procedure OnOffButtonClick(Sender: TObject);

private

  { Private declarations }

public

  { Public declarations }

end;

var

  MainForm: TMainForm;

implementation



---

```
{ $R *.DFM }
```

```
const
```

```
    numCheckBoxes = 6;
```

```
type
```

```
    PObjectArray = ^TObjectArray;
```

```
    TObjectArray =
```

```
        array[0..65520 div SizeOf(TObject)] of TObject;
```

```
var
```

```
    CheckBoxArray: PObjectArray; //Pointer to array of checkBoxes
```

```
procedure TMainForm.FormCreate(Sender: TObject);
```

```
begin
```

```
    GetMem(CheckBoxArray, numCheckBoxes * SizeOf(TObject));
```

```
    CheckBoxArray^[0] := CheckBox1;
```

```
    CheckBoxArray^[1] := CheckBox2;
```

```
    CheckBoxArray^[2] := CheckBox3;
```

```
    CheckBoxArray^[3] := CheckBox4;
```

```
    CheckBoxArray^[4] := CheckBox5;
```

```
    CheckBoxArray^[5] := CheckBox6;
```

```
end;
```

```
procedure TMainForm.FormDestroy(Sender: TObject);
```

```
begin
```

```
    FreeMem(CheckBoxArray, numCheckBoxes * SizeOf(TObject));
```

```
end;
```

```
procedure TMainForm.OnOffButtonClick(Sender: TObject);
```

```
var
```

```
    I: Integer;
```

```
begin
```

## الباب الثامن : تكوين الـ Lists

```
for I := 0 to numCheckBoxes - 1 do
  with CheckBoxArray^[I] as TCheckBox do
    Enabled := not Enabled;
end;

end.
```

### استخدام الـ TStringList class :

Delphi يقدم class اخرى وهى الـ TStringList التى تستخدم الـ TList object لإنشاء string lists . تنحدر الـ TStringList من الـ TStrings ، وهى الـ class التى تستخدمها الـ components لخصائص الـ string-list (سوف اشرح المزيد عن الـ TStrings فى الفصل القادم) . باستخدام الـ TStringList ، يمكنك إنشاء lists مرتبطة للـ strings والـ objects الأخرى ، ويمكنك تعيين تلك القوائم لخاصية الـ TStrings فى أى Delphi component object .

ان واحدة من اشهر استخدامات الـ TStringList هى قراءة وكتابة ملفات النص . فى الماضى البعيد ، لان الـ TStringList كانت تستخدم الـ Pascal strings ، لم تكن السطور اطول من ٢٥٥ رمزا . لم يعد هذا الحد موجوداً الآن ، فالـ Object Pascal Strings يمكن ان تكون بأى طول . توضيح القائمة (٨-٨) كيفية استخدام الـ TStringList لإنشاء ملف نص جديد . لتجربة الـ code ، قم بإنشاء مشروع وأضف إثنين من الـ Button objects على الـ form . وكذلك أضف الـ Memo component . أدخل العبارات من القائمة (٨-٨) فى الـ OnClick الخاص بالـ Button1 .

#### القائمة (٨-٨) : يوضح OnClick كيفية استخدام الـ TStringList لإنشاء ملف نص

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SL: TStringList;
begin
  SL := TStringList.Create;
  try
```

```

SL.Add('This is the first line');
SL.Add('This is the second line');
SL.Add('This is THE END');
SL.SaveToFile('Anyname.txt');
finally
    SL.Free;
end;
end;

```

يقوم البرنامج أولاً بإنشاء الـ TStringList يسمى SL. ثم يضيف بعض الـ strings الى القائمة، ويستدعى الـ SaveToFile اكتبها في ملف نص يسمى Anyname.txt. يقوم الـ procedure باستدعاء Free لحذف الـ string list object من الذاكرة. دائماً أمحو الـ objects التى تنشئها. ويضمن عبارة الـ try-finally-end ان الـ SL تم محوه، حتى لو كانت أى عبارة بعد الـ try تسبب exception.

توضح القائمة (٨-٩) كيفية قراءة ملف نص فى الـ TStringList. اذا كنت تنشئ المشروع، استخدم هذا الـ code فى الـ OnClick الخاص بالـ Button2. يقوم الـ procedure بإنشاء الـ object SL للـ TStringList، ثم يستدعى الـ LoadFromFile لقراءة سطور من الـ Anytext.txt فى object list. ويقوم بتعيين تلك النسخ من البيانات الى خاصية الـ Lines لـ object's Memo1، وهو object من نوع الـ TStrings.

لان الـ TStringList تنحدر من الـ TStrings، يمكنك تعيين أى object TStringList لخاصية الـ TStrings لـ object's component. يمكنك أيضاً تعيين ايه خاصية الـ TStrings لـ object TStringList.

**القائمة (٨-٩): يوضح OnClick هذا كيفية استخدام الـ object TStringList لقراءة ملف نص**

```

procedure TForm1.Button2Click(Sender: TObject);
var
    SL: TStringList;
begin

```

## الباب الثامن : تكوين ال Lists

```
SL := TStringList.Create;
try
    SL.LoadFromFile('Anyname.txt');
    Memo1.Lines := SL;
finally
    SL.Free;
end;
end;
```

توضح القائمة (٨-٩) كيفية قراءة ملف نص في TStringList object مستقل ، والذي قد تريد فعله لكثير من اغراض التعامل مع القوائم . ولكن ، يمكنك ببساطة اكثر ان تقوم بتحميل خاصية ال Lines ل Memo component باستدعاء ال LoadFromFile method الخاص بها . فهذه العبارة تفعل كل ما تفعله القائمة (٨-٩) :

```
Memo1.Lines.LoadFromFile('Anyname.txt');
```

استخدام ال AddStrings لإلحاق string lists متعددة ال TStringList object أو خاصية ال TStringList . على سبيل المثال اذا لديك ثلاثة من ال TStringList - T1 ، T2 ، T3- يمكنك إدخالها في خاصية ال Lines ل Memo object بواسطة العبارة التالية . يحل التعيين الاول محل نص ال Memo ب strings من ال T1 . والتعيين الآخرين يلحقا القائمتين الاخرتين :

```
with Memo1, Lines do
begin
    Lines := T1;
    AddStrings(T2);
    AddStrings(T3);
end;
```

TStringList لابد من تحديد قيمة لها ، ليس فقط string lists ، ولكن ايضاً لإنشاء روابط بين ال strings وال objects . يوضح الفصل القادم كيفية استخدام هذه الإمكانية لعرض bitmaps في ال ListBox control .

### استخدام ال TString class :

تعتبر ال abstract class ، وهذا يعنى ان البرامج لا تستطيع إنشاء objects من هذا النوع . قد تستخدم ال TString فقط كخاصية object component .

مثلاً، يمكنك تخزين قائمة من الـ strings والـ objects الأخرى فى خاصية الـ Items التابعة لـ ListBox، وهو TStrings object. لإنشاء list objects الخاصة بك، استخدم الـ TList والـ TStringList classes كما هو موضح فى الفصول السابقة.

لان الـ TStringList تنحدر من الـ TStrings، فإن المعلومات الواردة فى هذا الباب تنطبق على كلتا الـ classes. فأى شئ يمكنك فعله مع خاصية الـ TStrings، يمكنك فعله أيضاً مع الـ TStringList object.

### استخدام الـ StringGrid :

يعد الـ StringGrid مربع list box متقدم قادراً على ربط الـ string والـ object الأخرى مثل الـ bitmap. يمكن ان يعرض الـ StringGrid جداول لبيانات الـ strings، كلاً منها مرتبط بـ object آخر.

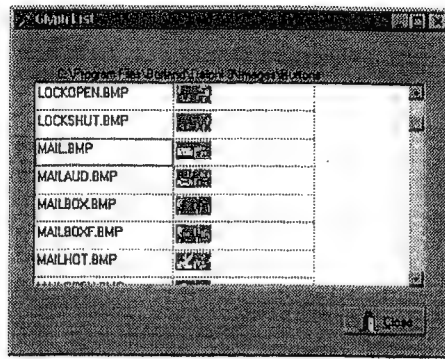
**على القرص المدمج:** يوضح البرنامج التالى، الـ GlyphList كيفية استخدام الـ StringGrid لإنشاء array of strings. والـ bitmaps يعرض البرنامج كل glyph bitmaps الخاصة بـ Delphi واسماء الملفات الخاصة بها فى owner-draw string grid object. (انظر الباب السادس لمزيد من المعلومات عن الـ glyphs). والمصطلح owner-draw يعنى أن البرنامج، بدلاً من الـ Windows، يتولى مسئولية رسم كل عنصر فى خلية الـ StringGrid. يوضح شكل (٨-٦) عرض الـ GlyphList. توجد الـ source code فى القائمة (٨-١٠). يمكنك أن تجد ملفات البرنامج على القرص المدمج فى دليل الـ Source\GlyphList. لقد أدخلت تعليقات فى الـ code لتوضيح كثير من العبارات، التى تستخدم بعض التقنيات لم يتم تقديمها بعد.



**ملحوظة:** اذا لم تقم بتركيب Delphi فى الأدلة الافتراضية على محرك الـ C:، قم بتعديل string اسم المسار فى ثابت الـ glyphPath، الموضوع بعد كلمة التنفيذ الرئيسية للـ unit مباشرة.



## الباب الثامن : تكوين الـ Lists



شكل (٨-٦) : يعرض الـ GlyphList الـ owner-draw ListBox لكل bitmaps الموجودة مع Delphi

القائمة (٨-١٠) : Glyphlst\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics,  
Controls,
```

```
Forms, Dialogs, StdCtrls, Buttons, Grids;
```

```
type
```

```
TMainForm = class(TForm)
```

```
PathLabel: TLabel;
```

```
BitBtn1: TBitBtn;
```

```
GlyphList: TStringGrid;
```

```
procedure FormCreate(Sender: TObject);
```

```
procedure FormClose(Sender: TObject;
```

```
var Action: TCloseAction);
```

```
procedure GlyphListDrawCell(Sender: TObject;  
Col, Row: Integer; Rect: TRect; State: TGridDrawState);  
private
```

```
{ Private declarations }
```

```

public
    { Public declarations }
end;

var
    MainForm: TMainForm;

implementation
{$R *.DFM}

const glyphPath =
    'C:\Program Files\Borland\Delphi 4\Images\Buttons';

// Reads all glyph bitmaps and stores their filenames and
// images in the GlyphList TStringGrid object.
procedure TMainForm.FormCreate(Sender: TObject);
var
    SearchRec: TSearchRec; { Directory scan result record }
    K: Integer;           { while-loop control variable }
    Bitmap: TBitmap;      { Holds bitmaps. Do not Free! }
    Index: Integer;       { TStringGrid cell index }
begin
    Show; { Make form visible while loading bitmaps }
    Screen.Cursor := crHourGlass; { Show hourglass cursor }
    Index := 0;
    try
        PathLabel.Caption := glyphPath; { Show path above
        ListBox }
        { Start scan }
        K := FindFirst(glyphPath + '\*.*', faAnyFile,
        SearchRec);
        try
            while K = 0 do { Scan directory for file names }
            begin

```

```

if SearchRec.Name[1] <> '.' then {No '.' or '..' paths}
begin
    Bitmap := TBitmap.Create; { Create bitmap object }
    try { Get bitmap and load from list }
        Bitmap.LoadFromFile(glyphPath + '\'
            + SearchRec.Name);
        if Index = GlyphList.RowCount then // Expand
list
            GlyphList.RowCount := Index + 1;
            GlyphList.Cells[0, Index] := SearchRec.Name;//
Name
            GlyphList.Objects[1, Index] := Bitmap; //
Bitmap
        inc(Index);
        except
            Bitmap.Free; { Executed if ANYTHING goes wrong }
            raise; { Pass any exceptions up call
chain }
        end;
    end;
    K := FindNext(SearchRec); { Continue directory scan
}
end;
finally
    FindClose(SearchRec);
end;
finally
    Screen.Cursor := crDefault; { Restore normal cursor }
end;
end;

// Frees memory occupied by all glyph bitmaps. The
TStringGrid
// object does NOT do this automatically.
procedure TMainForm.FormClose(Sender: TObject;

```



```

var Action: TCloseAction);
var
  I: Integer;
begin
  for I := 0 to GlyphList.RowCount - 1 do
    TBitmap(GlyphList.Objects[0, I]).Free;
end;

// Draw each glyph bitmap. The TStringGrid object is smart
// enough to draw its filename text objects with no further
// help. This code draws only the bitmaps in the second
// column.
procedure TMainForm.GlyphListDrawCell(Sender: TObject;
  Col, Row: Integer; Rect: TRect; State: TGridDrawState);
var
  Bitmap: TBitmap;
begin
  if col = 1 then // Be sure to refer to the bitmap column
  begin
    { Get bitmap object }
    Bitmap := TBitmap(GlyphList.Objects[1, Row]);
    if Bitmap <> nil then
      begin { Draw bitmap in column cell }
        GlyphList.Canvas.BrushCopy(
          Bounds(Rect.Left + 2, Rect.Top + 2, Bitmap.Width,
            Bitmap.Height), Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);
        { The preceding clRed argument gives the
        transparent
        glyph substance. Change this to any solid color,
        or
        change it to 0 to see why this is necessary. }
      end;
    end;

```

## الباب الثامن : تكوين الـ Lists

```
end;  
end;
```

```
end.
```

يعرض الـ GlyphList بعض التقنيات الأساسية التي يمكنك استخدامها في جميع التطبيقات مع StringGrid object. تحتوي الـ Form الرئيسية على ثلاث components فقط : Label1 ، BitBtn1 ، و GlyphList ، من الـ TStringGrid class . (في العادي كنت سأجعل اسم هذا الـ object GlyphStringGrid ، ولكنني تركت الاسم GlyphList لأن النسخة الأصلية من هذا البرنامج تستخدم List object بدلاً من الـ component الأكثر تعقيداً (StringGrid).

لقد قمت بتعديل خصائص GlyphList object لإنشاء قائمة ذات عمودين . يعرض البرنامج أسماء ملف glyph في العمود الأول . يوضح العمود الثاني الـ bitmap المرتبطة بكل glyph . لبرمجة الـ StringGrid الخاص بالتطبيق ، قمت بتغيير هذه الخصائص الهامة :

● **ColCount = 2** : هذا هو عدد الأعمدة والذي يمكن تغييره في وقت التشغيل .

● **DefaultColWidth = 128** : يحدد عرض كل عمود . لأن هذه هي شبكة صفوف وأعمدة ، فكل خلية لها نفس العرض .

● **FixedCols = 0; FixedRows = 0** : هذه المواصفات تزيل رؤوس أعمدة و صفوف الـ Grid التي تراها على الـ StringGrid الافتراضي . هذه القيم محددة في العادي بـ 1 لتوفر عموداً واحداً و صفراً واحداً لمعلومات الرأس مثل الـ label الخاص للعمود .

قد تريد عادة ان تتولى رسم كل خلية في الـ StringGrid . إنك تفعل هذا بإنشاء الـ OnDrawCell الخاص بالـ StringGrid . وهذا يؤدي الى إنشاء ما هو معروف باسم الـ owner drawn control . لكل خلية في الـ Grid ، يقوم Delphi باستدعاء الـ event handler الذي يرسم ما تريده داخل حدود الخلية .

وهناك خاصية أخرى، وهى الـ `DefaultDrawing`، تؤثر على كيفية عمل الـ `StringGrid` والـ `event handler` الخاص بك معاً، كما هو موضح هنا:

● **DefaultDrawing = True**؛ ترسم الـ `StringGrid` خلفية الخلية وتعرض ايه بيانات نص مرتبط بالخلية قبل استدعاء `OnDrawCell`. بعد إدخال الـ `event handler`، يرسم Delphi مستطيل حول الخلية التى تم اختيارها حالياً.

● **DefaultDrawing = False**؛ يعتبر `OnDrawCell` الخاص بك مسئول عن رسم كل خلفيات وواجهات كل خلية.

يستخدم برنامج الـ `GlyphList` الخيار الأول: `DefaultDrawing = True`. يملأ `OnDrawCell` الخاص بالـ `StringGrid` كل زوج من الخلايا باسم ملف الـ `glyph's` و `bitmap` المتعلق به. ان كل استدعاء `GlyphListDrawCell` `procedure` يحتوى على ثلاث معلومات هامة تم تمريرها كـ `parameters`.

● **Col**؛ رقم عمود الخلية.

● **Row**؛ رقم صف الخلية.

● **Rect**؛ حدود الرسم لمستطيل الخلية.

ان العمود والصف الأولين كل منهما يحمل الرقم صفر. فى هذه الحالة، لا يوجد لدينا قمم أعمدة أو صفوف- ولكن اذا استخدمتها، تذكر انك تحتاج ان ترسم هذه الخلايا ايضاً. ان `GlyphList` لا يستخدم الرؤوس.

هناك خاصيتين تحملان الـ `StringGrid's string` وبيانات الـ `object` المرتبط به لكل خلية. يتم تعريف خاصية الـ `Cells` كما يلى:

`property Cells[ACol, ARow: Integer]: string;`

تعتبر كل خلية `string object`، بسيط وخالٍ. يتم تعريف خاصية الـ `Objects` بأسلوب مشابه، ولكن يمكنها ان تحمل أى نوع من الـ `class object`، طالما انها تنحدر من الـ `TObject`:

`property Objects [ACol, ARow: Integer]: TObject;`

## الباب الثامن : تكوين الـ Lists

معاً، توفر خاصتى الـ Cells والـ Objects الوصول الى الـ string والبيانات الاخرى لكل خلية من الـ StringGrid. ان عرض هذه البيانات هو مجرد توصيلها من خلال هذه الـ arrays. لان بيانات النص يتم رسمها تلقائياً، يقوم الـ event handler الخاص بنا أولاً بالتأكد من ان الـ Col يساوى 1. اذا كان كذلك، فإن الـ StringGrid يطلب ان يرسم التطبيق بيانات جرافيكية. وهذه لبيانات مخزنة فى خاصية الـ Objects للـ Grid. بفرض ان الـ Bitmap هو object مؤقت الـ TBitmap class، فهذه العبارة:

```
Bitmap := TBitmap(GlyphList.Objects[1, Row]);
```

تحصل على reference للـ Bitmap object المخزن فى العمود 1، والصف المشار إليه. بعد الحصول على هذا reference، يرسمه البرنامج باستدعاء الـ BrushCopy procedure الخاص بالـ StringGrid للـ Canvas التابع للـ object's. اختبر هذه العبارة فى القائمة، ولاحظ كيف يتم استخدام الـ Rect parameter لوضع الصورة داخل كل خلية. ان القيمة المتغيرة clRed فى هذه العبارة الـ glyph شفافاً.

**ملحوظة:** اذا كنت على دراية ببرمجة الـ Windows، فإن الـ Canvas يعادل الـ device context لا تهتم اذا لم تكن تعرف ما هى الـ device context سوف تعرف عنهم وعن الـ Canvas الكثير فى الباب ١٣.



للإعداد لرسم كل Grid، يستخدم البرنامج الـ OnCreate لنافذة الـ form الرئيسية. ولأن تحميل كل ملف الـ glyph bitmap قد يستغرق وقتاً، يبدأ الـ FormCreate procedure بتحديد الـ Screen cursor بـ crHourGlass. ثم يستدع الـ FindFirst لوضع أول ملف مناسب. يتم إنشاء الـ Bitmap object بهذه العبارة.

```
Bitmap := TBitmap.Create;
```

ثم يتم استخدام هذا الـ object لتحميل الـ bitmaps باستخدام العبارة:

```
Bitmap.LoadFromFile(glyphpath + '\' + SearchRec.Name;
```

وتقوم خاصية الـ Name فى متغير الـ SearchRec بالاحتفاظ بالنتائج من استدعاء الـ FindFirst. لتوفير مكاناً فى الـ Grid للاحتفاظ بـ bitmap واسم ملف آخرين، ينفذ البرنامج هذا الـ code:

```
if Index = GlyphList.RowCount then
  GlyphList.RowCount := Index + 1;
```

إن زيادة الـ RowCount بواحد يزيد كمية المعلومات التي يمكن أن تحملها وتعرضها الـ StringGrid. بعد ذلك تقوم عبارتان إضافيتان بتعيين bitmap واسم الملف لخلية الـ Grid :

```
GlyphList.Cells[0, Index] := SearchRec.Name;
GlyphList.Objects[1, Index] := Bitmap;
```

تحمل خاصية الـ Cells بيانات string كل خلية، والتي، في هذا المثال، تم تعيين اسم ملف الـ glyph لها من سجل الـ SearchRec. يتم استخدام خاصية الـ Objects اختياريًا لربط بيانات أخرى بـ string كل خلية - في هذه الحالة، يعين البرنامج glyph bitmap يكون قد تم تحميلها من القرص.

من المهم أن تفهم أن التعيين إلى الـ Objects يربط Bitmap بهذه الخلية من الـ Grid. ولا تنسخ العبارة الـ Bitmap إلى موضع جديدة لهذا السبب، يجب إنشاء Bitmap object جديد باستدعاء TBitmap.Create لكل object جديد تم تعيينه في الـ Objects array في الـ StringGrid.

تتواجد كل هذه الـ code داخل الـ try-except block، لذا إذا حدث أي خطأ - يحدث خطأ قرص، مثلاً، أو تنتهي ذاكرة البرنامج - يتم محو الـ Bitmap object الحالي ويتم تمرير الـ exception لمستدع الـ event handler. إذا سار كل شيء على ما يرام بعدد معالجة كل glyph، يستدع البرنامج FindNext. استدع الـ FindFirst لتبدأ العملية ثم استدع الـ FindNext تكراراً حتى تدخل الـ function قيمة غير صفرية، مما يشير إلى أن بحث أسماء الملف قد تم.

لاحظ أن الـ event handler يستخدم عبارتي finally في نهايته، كلاً منها مرتبطة بـ try block. تنهي الـ FindClose بحث أسماء الملف، وتطلق Resources النظام الموجودة الـ SearchRec object. من الصواب أن تستدع الـ FindClose داخل الـ finally block لضمان أن هذا الـ procedure تم استدعاءه حتى إذا حدث الـ exception في مكان ما في الـ try block المرتبط به.

## الباب الثامن : تكوين الـ Lists

وهناك فكرة صائبة أخرى، وإن كانت أقل أهمية، وهى أن تعيد تحديد الـ screen cursor بـ crDefault فى عبارة finally. هذا يضمن أن cursor دخلت فى شكلها العادى حتى اذا حدث خطأ فى الـ procedure.

إن الـ StringGrid غير قادر على محو بيانات object تم تعيينها الى الـ Objects array. إنه يتحكم تلقائياً فى الذاكرة التى تستخدمها خاصية الـ Cells، ولكنها مسئوليتك أن تفعل هذا البنود مثل bitmaps المعنية للـ Objects. وهذا ضرورى لأن الـ StringGrid لا يعرف أى نوع من البيانات قمت بتعيينه للـ objects. بالرغم من أنه يتم استخدامه بشكل نموذجى للإشارة الى الـ Bitmap objects، فمن الممكن تعيين أى نوع من objects لهذا الـ array. وهذا يأتى على حساب أن يجعلك مسئولاً عن تحرير أية ذاكرة مشغولة بهذا الـ objects.

والمكان الجيد لمسح الـ StringGrid object هو فى OnClose event handler الخاص بالـ form. فى الـ GlyphList، مثلاً، محور الـ loop التالية كل الـ Bitmap objects المخزنة كـ references فى الـ Objects array:

```
var
  I: Integer;
begin
  for I := 0 to GlyphList.RowCount - 1 do
    TBitmap(GlyphList.Objects[0, I]).Free;
end;
```

تشير خاصية الـ RowCount الى كم العناصر التى تم تخزينها فى الـ StringGrid. باستخدام هذه القيمة، فمن المستحسن إنشاء loop for تستدعى TBitmap.Free لكل object Bitmap مذكور لـ referenced من قبل الـ Objects array.

**على القرص المدمج:** لأغلب المعلومات عن قوائم الـ string، اختبر أيضاً التطبيق ToDo على القرص المدمج فى دليل الـ Source\ToDo. يظهر هذا البرنامج تقنيات TStringList متنوعة. على سبيل المثال، توضح القائمة (٨-١١) الـ procedure الذى يستخدمه الـ ToDo لنقل عنصر مختار من احد الـ ListBox الى



آخر. لاحظ ال TCustomListBox parameter . يمكنك استخدام ال TCustomListBox's procedure Type ، ولكن استخدام ال TCustomListBox procedure ان ال procedure يعمل لأية objects اخرى من classes مشتقة من ال TCustomListBox class .

القائمة (٨-١١)؛ يستخدم ToDo هذا ال procedure  
لنقل العناصر المختارة من ListBox الى آخر

```
procedure TMainForm.MoveSelected(List: TCustomListBox;
  Items: TStrings);
var
  I: Integer;
begin
  for I := List.Items.Count _ 1 downto 0 do
    if List.Selected[I] then
      begin
        Items.AddObject(List.Items[I], List.Items.Objects[I]);
        List.Items.Delete(I);
      end;
  FileDirty := True;
end;
```

يستخدم برنامج ال ToDo ال MoveSelected باستخدام عبارة مثل التالية،  
والتي تنقل المداخل المختارة بين ال ، ListBox objects من SrcList الى DstList :

MoveSelected(DstList, SrcList.Items);

فى ال MoveSelected (راجع القائمة)، يعتبر ال List parameter هو  
ال ListBox الخاص بالمسافة؛ وال Items parameter هو ال TStrings source .  
وتقوم ال for loop بقياس ال List من اسفل الى اعلى (لاحظ استخدام الكلمة  
الرئيسية downto الخاصة بال Pascal). يجب ان يكون القياس فى هذا الاتجاه لان  
أى حذف فى البند N يغير فهارس البنود .... N+1.

اذا تم اختيار عنصر قائمة، ينسخ ال AddObject ال List's string وأى  
object مرتبط به الى ال Items parameter، من نوع ال TStrings . يحذف

## الباب الثامن : تكوين الـ Lists

البرنامج بعد ذلك البند الافتراضي . ان استدعاء الـ Delete يحذف الـ string المشار إليه والذي تم تمريره من خلال الـ (I) index argument ، وكذلك يحدد الـ Objects[I] بلا شيء . لذا ، فإن استدعاء . واحدًا للـ Delete يزيل string الـ list's وأي object مرتبط به . ولكن ، إنه لا يتخلص من ذاكرة هذا الـ object - إنك يجب ان تفعل هذا دائماً باستدعاء الـ Free .

### افكار للمستخدم الخبير

• انظر مزيداً من المعلومات على رسائل الـ lb\_ في Windows API online الخاصة بـ Delphi . ان استدعاء الـ Perform لإرسال رسالة الـ scroll bar الافقي في list box مثلاً - يعتبر بشكل عام ، ولكن ليس بالضبط ، مساو لاستدعاء الـ Windows SendMessage function . يقوم الـ Perform باستدعاء الـ VCL WndProc ؛ يُستدعى الـ SendMessage للنافذة العلوية المرتبطة مع window . handle

• يمكنك أيضاً استخدام الـ TStringList (أو خاصية الـ TStringList لإنشاء strings lists و objects متعلقة . ان التقنيات تشبه تلك الموضحة في تطبيق الـ GlyphList ، والتي تستخدم الـ StringGrid .

### المشروعات التي يمكنك تجربتها

- (٨-١) : اكتب تطبيقاً يقدم الـ ListBox لفرز ملف نص .
- (٨-٢) : اكتب code تعكس ترتيب المداخلات في الـ ComboBox object أو الـ ListBox .
- (٨-٣) : اكتب procedure يحمل اسماء ملف دليل في الـ TStringList object .
- (٨-٤) : قم بتحسين تطبيق الـ GlyphList باضافة قائمة File ذات اوامر لتغيير الادلة . ملحوظة : لمنع البرنامج من محاولة تحميل صور



الـ Bitmaps ضخمة فى الـ ListBox ، فقد تريد إضافة code ترفض الـ Bitmap objects الأكبر من الحجم المحدد سابقاً .

(٥-٨) : متقدم . قم بإنشاء ListBox owner-draw يعرض string لها عدة أنماط من الـ Font . (ملحوظة : ان بعض الـ Fonts رمزية ، ويجب ان تعرض اسم الـ Font منفصلاً عن الـ string الخاص به) .

### ملخص:

- ان الـ ListBox والـ ComboBox بـ Delphi لا تعتبر قيمة لإضافة قوائم قابلة للاختيار الى واجهات التطبيق للتطبيق . يمكنك الاختيار من بين عدد من الانماط لإنشاء قوائم متعددة الاختيارات والسماح للمستخدمين بادخال بيانات جديدة فى نافذة الـ ComboBox's edit .

- يقدم Delphi أيضاً classes صنع القوائم المرتبطة وهى TList ، TStringList و TStrings . استخدم الـ TList والـ TStringList لإنشاء list objects مستقلة فى الذاكرة . لا يمكنك إنشاء TStrings objects - الـ components فقط يمكنها استخدام الـ TStrings كـ class type's property .

- ان كل الـ classes فى Delphi انحدر من الـ TObject . هذا يعنى ان كل class objects فى تطبيقات Delphi مرتبطة .

- لان الـ TStringList تنحدر من الـ TStrings ، فأى شئ يمكنك فعله بخاصية الـ TStrings (مثل خاصية الـ Items للـ ListBox) . يمكنك ان تفعله مع TStringList مستقل .

- StringGrid يقدم الـ general-purpose row-and-column ، والذي يمكنه ان يحمل string والمعلومات المتعلقة بكل خلية فى الـ Grid . بشكل نموذجي ، تعتبر المعلومات المتعلقة بالـ Bitmap object ، ولكنها يمكن ان تكون أى object آخر من class تنحدر من الـ TObject . ان خصائص الـ Objects والـ Cells للـ StringGrid بـ arrays ثنائى الابعاد ، والذي يمكن للبرنامج من خلاله ان يتوصل لمحتويات الـ Grid .

## الباب الثامن: تكوين الـ Lists

• يوفر الـ TStrings objects أو خاصية الـ TStrings الوصول الى بيانات الـ string من خلال خاصية Items، أو من خلال الـ Items.Strings الذي يساويه وتعتبر بيانات الـ object المرتبطة متاحة من خلال الـ Objects array.

• انها مسئوليتك ان تمحو الـ object التي تنشئها. وهذا يعتبر مهماً بشكل خاص عند استخدام الـ Objects array للـ StringGrid object. ان افضل مكان لمحو objects الخاصة بالـ Grid هو عادة في الـ OnClose الخاص بالـ form.

• تقوم الـ TList والـ TStringList تلقائياً بتحرير بيانات الـ string فقط. قمت بإضافة objects اخرى في قائمة، يجب ان تمحو تلك الـ objects باستدعاء الـ Free.

يمكنك ان تحدد عشرات الاستخدامات لـ string lists في الـ component المؤسسة على نص والخاصة بـ Delphi مثل الـ Memo، الـ Notebook، والـ TabbedNotebook، والتي تقابلها في الباب التالي.



# الباب التاسع

## العمل مع Single-Line Strings

محتويات هذا الباب:

• Components

• Character strings

• Built-in text dialogs

• Single-line text components

في العصر الماضي للحاسوب، كنا نحن المبرمجون محظوظون إذا كانت تطبيقاتنا بها حروف كبيرة وحروف صغيرة زائد القليل من رموز علامات الترقيم المعيارية. ولحسن الحظ، كان باستطاعتنا الاختيار من بين مجموعة كبيرة من الالوان- مادامت لا تخرج عن الأخضر والبرتقالي. اليوم، مع الـ Windows واجهات تطبيق المستخدم الجرافيكية الأخرى، فإن الـ TrueType Fonts والشاشات كاملة الالوان تخلق فرص لا حدود لها لمصممي واجهات التطبيق، ولكن كاحدى النتائج، فقد واجه المبرمجون مجموعة جديدة تماماً من القواعد واللوائح للعمل مع الـ character strings و text objects أخرى.

يختبر هذا الباب single-line strings و text components من جانبيين- التخزين الداخلى والعرض الخارجى. ان إنشاء واجهة تطبيق ناجحة لنظام يعمل بالجرافيك يتطلب الآن الانتباه الى العديد من تقنيات عرض وتخزين الـ strings الجديدة.

### : Components

فيما يلى الـ Delphi's components للعمل مع الـ single-line strings :

• **Edit**، أحد ال controls ذات المظهر البسيط فى ل Windows ، ولكنه أيضاً واحداً من الاكثر تقلباً، فال Edit component يمكن ان يتعامل مع اغلب مهام single-line . يقوم ال Edit تلقائياً بدعم عمليات القطع والنسخ واللصق ، وله خصائص التعامل مع الفأرة ولوحة المفاتيح المركبة داخلياً. ال Palette : Standard .

• **Label**، لقد استخدمت ال Label component من قبل فى الابواب السابقة ، ولكن ال Labels لها استخدامات اخرى قد لا تكون واضحة . على سبيل المثال ، يستطيع ال Labels ان يحدد hot key ل controls اخرى لا تؤيد اختيارات لوحة المفاتيح . ال Palette : Standard .

• **MaskEdit**، قد تعتقد ان هذا ال component هو Edit box ذا عقول . استخدم ال MaskEdit لإنشاء حقول صالحة لإدخال بيانات- على سبيل المثال ، مربع ادخال رقم الهاتف يتطلب عوامل لإدخال ارقام فى الاماكن المحددة . Additional : Palette .

## : Character Strings

يقدم Delphi مجموعة من ال string operators وال functions وال procedures . قبل إلقاء الضوء على ال text components وال string ، إنك تحتاج ان تفهم جيداً أثنان من ال string formats بال Delphi وكذلك ال functions وال operators الفصول التالية تقدم بعض المعلومات كخلفية عن نوع بيانات ال string الخاص ب Delphi .

## : Types of strings

عندما انتقل Delphi 2 الى ساحة نظام تشغيل ال ٣٢ بت ، تم إجراء تحويل طويل لنوع بيانات ال string الخاص بال Pascal القيمة . فى الماضى ، كان ال string عبارة عن array يصل الى ٢٥٦ رمزاً ذا ٨ بت ، والبايت الأول فى هذا ال array طول ال string بالرموز . ويسمى ب Length-Byte لا يزال متاحاً ك ShortString فى ال Object Pascal . والسبب الوحيد لاستخدام هذا ال string حالياً هو للتلائم الخلفى .

## الباب التاسع : العمل مع Single-Line Strings

والجديد فى ال Object Pascal هو نوع بيانات للـ AnsiString . مع هذا النوع ، من الممكن ان توجد string ذات طول غير محدد .

فى اغلب الحالات ، تعتبر الـ string التى تقوم بإنشاءها من هذا النوع .  
pascal's string data type تلقائياً يصبح من نوع هذا الـ String .

**ملحوظة:** ان الاحرف الصغيرة تشير الى أى نوع من أنواع الـ string فى حين ان بدء الكلمة بحرف كبيرة يشير الى الـ Object Pascal's data type بشكل خاص .

Note

يمكنك تغيير نوع بيانات الـ string لتمثيل أى من نوعى الـ string باستخدام امر الـ \$H compiler . فى الحالة الافتراضية ، { \$H+ } ، يعتبر الـ string هو نفسه الـ AnsiString . فى حالة الـ { \$H- } ، يرجع الـ string الى طول البايت القديمة . من الافضل فى معظم الحالات ان تستخدم التعليق الافتراضى وان تستغل نوع بيانات الـ AnsiString الجديد . استخدم التحديد المقابل فقط اذا يجب عليك ان تجرى عملية الـ compile للـ code وتتوقع ان يكون الـ string هو طول البايت القديمة .

ونوع آخر من الـ string هو الـ WideString ، والذي يمكن ان يخزن رموزاً للغات تحتاج متغيرات ذات حروف أكبر . تقوم الـ WideString objects بتخزين الـ Unicode ، رموز ١٦ بت ، والتي تسمح بنص فى لغة معينة ان يكون اسهل نقلاً (وفى بعض الحالات يتم عرضه دون غموض) الى مجموعة من نظم الحاسوب .

**ملحوظة:** تستخدم الـ OLE ذات الـ ٣٢ بت الـ WideString لكل بيانات الـ string . وبالنسبة للتحكم فى الذاكرة ، يتم استخدام اثنان من الـ WideString objects والـ AnsiString بطريقة متشابهة ، كلاهما يمكن ان ينشئ متغيرات string ذات اطوال غير محددة .

Note

### String طويلة وقصيرة:

ان string يسهل إنشاءها واستخدامها . حدد متغير string باستخدام تعريف الـ var ، إما فى قطاع الـ unit's implementation ، أو فى الـ procedure أو الـ function :

var

S: String;

هذا ينشئ string object خال، والذي يمكن ان يحمل من صفر الى أى عدد من الرموز. ويعرف ال object أيضاً dynamic string لان حجمه بتغير ديناميكياً طبقاً لطول بيانات ال string الفعلية التى تم تعيينها لل object. بعد تعريف متغير ال string ، يمكنك تعيين string حرفى لل S :

S := 'Click!';

يمكن متغير الى S الخاصية ال Caption ل control مثل ال Button. هذه العبارة، على سبيل المثال، تغير نص ال Button object الى بيانات الرمز فى متغير ال string :

Button1.Caption := S;

يمكنك تنفيذ عمليات مشابهة لأى خاصية string. يمكنك ايضاً استخدام أى String object ك array للرموز. ان الرمز الأول فى ال string يحمل ال index 1، لذا فهذا التعيين :

S[2] := 'X';

يعين X للرمز الثانى لمتغير ال string S. الرموز الفردية هى من نوع بيانات ال Char، وهى مساوية لل AnsiChar (رموز ذات ٨ بت). فى أغلب البرامج، يمكنك استخدام متغيرات ال Char و ال string دون الاهتمام بانواعها. ولكن، اذا كان يجب عليك استخدام رموزاً عريضة، يجب ان تعرف string objects الخاص بك من نوع ال WideString ومتغيرات رموزك من نوع ال WideChar. ان Delphi لا يقوم حالياً بالانتقال التلقائى بين ال ٨ بت العادية و string الرموز العريضة.

يمكنك ان تستخدم string طول البايت الافتراضى الخاص بال Object Pascal، اذا كنت تفضل هذا. يمكنك إنشاء واحداً من هذه الاشياء المعروفة الآن ب ShortString، بطريقتين اساسيتين. استخدم اسم نوع البيانات هذا فى التعريف مثل :

var

SS: ShortString;

## الباب التاسع : العمل مع Single-Line Strings

هذا ينشئ متغير SS القادر على حمل ٢٥٥ رمزاً. ولأن هذا عادة مضبوطة للوقت (ان كل ال strings القصيرة هي كلها ذات هذا الحد الأعلى من الطول)، فالطريقة الأخرى والأفضل عادة للإنشاء هي بتحديد حد أعلى للطول بين قوسين:

var

SS: String[4];

لاحظ انك تستخدم تعريف ال String، وليس ال ShortString، لإنشاء ShortString object ذا حجم محدد. كما هو مقرر هنا، ان حاول ال (SS) هو خمسة بايت - اربعة بايت لعدد رموز، وواحدة لطوله المخزن كقيمة البايت في ال SS[0].

**ملحوظة:** يمكن ان تدفع ال Short string الأداء سلباً - وهو سبب آخر جيد لعدم استخدامها. مثلاً، ان كل سبل استغلال ال strings الخاصة بـ Delphi تقبل ال strings parameters الطويلة. اذا قامت بتمرير strings قصيرة الى هذه السبل، فإنها تتغير داخلياً من والى ال strings الطويلة، مما يهدر الوقت والذاكرة. لأفضل أداء، استخدم نوع بيانات ال string الخاصة بال Pascal، واستخدم ال ShortStrings فقط عند الحاجة الماسة.

يمكنك مقارنة ال String objects ابجدياً باستخدام العوامل المنطقية المعتادة: <، >، <=، >=، و. على سبيل المثال، ال code التالي توضيح كيفية مقارنة إثنين من ال String objects ابجدياً:

var

S1, S2: String;

begin

S1 := 'ABCDEFGH';

S2 := 'abcdefgh';

if (S1 > S2) then

ShowMessage('S1 > S2')

else

ShowMessage('S1 <= S2');

end;

ملحوظة لمبرمجي ال C وال C++ : من الممكن ايضاً استخدام ال Object Pascal لتحديد null-terminated string كـ array لل Char. هذا ينشئ نوع



بيانات مساو لـ null-terminated string فى برمجة لغات مثل الـ C++ .  
على سبيل المثال، ما يلى يحدد string من ٦٤ رمزاً زائد بايت واحد يحتفظ به  
للـ terminating null :

var

S: array[0 .. 64] of Char;

بالرغم من ان هذا مسموح به، الا انه من افضل استخدام الـ strings الخاصة  
بالـ Pascal بدلا من الـ null-terminated character arrays . ان الـ String  
function بالـ Delphi فعالة للغاية، اسهل فى الاستخدام، ويمكن الاعتماد عليها  
أكثر من كتابة functions خاص بك- حتى اذا كنت على دراية بالـ C والـ C++  
والـ null-terminated strings والـ string pointers . كقاعدة عامة، إنك تحتاج  
الـ null-terminated strings لتمرير pointers الى الـ Windows functions  
أو للإستخدام مع functions و library procedures التى تتطلب هذا النوع من  
البيانات .

**ملحوظة:** ان الـ strings الطويلة تكون دائماً الـ null-terminated  
strings بالرغم من أنه لا يوجد function أو procedure يتطلب هذا  
كقاعدة . ولأنها null-terminated، فالـ strings الطويلة تعد  
type-cast متماشياً مع الـ PChar pointer .

Note

## : Built-In Text Dialogs

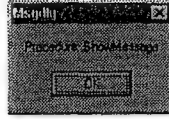
اذا لم تكن قد إكتشفت string input and output dialogs وفى  
الـ Delphi، فإننى اذكرها هنا إنها ادوات سهلة الاستخدام لعرض رسائل وحث  
المستخدمين على الادخال جرب هذه- ان text single-line utilities لها  
عشرات الاستخدامات .

## : Displaying text messages

يقدم Delphi خمسة message dialogs، واحدة منها من المؤكد ان تفى  
باحتياجاتك . توضح الأشكال من (٩-١) الى (٩-٥) الـ dialogs الخمس بترتيب  
درجة التعقيد .

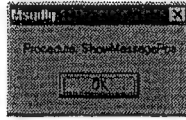
## الباب التاسع : العمل مع Single-Line Strings

• يعرض ال ShowMessage procedure الرسائل الثابتة فقط ، كما هو موضح في شكل (١-٩) .



شكل (١-٩) : يعرض ال ShowMessage  
ال string في dialog box مع زر OK

• ان ال ShowMessagePos procedure هو نفسه ال ShowMessage ولكن باحداثيات ال x- و ال y- التي تحدد موضع dialog box بالنسبة للشاشة [انظر شكل (٢-٩) ] .



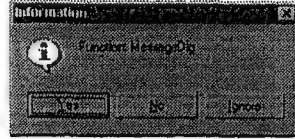
شكل (٢-٩) : تحتوي ال ShowMessagePos dialogs  
على احداثيات x- و y-

• يقوم MessageBox procedure باحتواء ال Windows API MessageBox function . في شكل (٣-٩) في ال Windows 3.1 ، تعرض ال functions ، displays غير ال 3D ، قد يتم الاسلوب . للحصول على نتائج افضل ، استخدم واحدة من functions التالية بدلاً منه . في ال Windows 95 ، ينتج ال MessageBox نفس النتائج البصرية كالـ MessageDlg وال MessageDlgPos . (جميع ال controls في ال Windows 95 ، NT ، وال Windows 98 لها مظهر 3D ) .



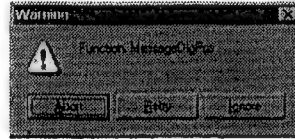
شكل (٣-٩) : ال MessageBox dialog يحتوي  
ال Windows API MessageBox function

• يقدم ال MessageDlg procedure مجموعة واسعة من الخيارات لعرض الرسائل والحث على إجابة الاسئلة. يمكنك اختيار ايقونة [مثلاً ، دائرة المعلومات الموضحة فى شكل (٩-٤)] ، ويمكنك اختيار انواع وعدد من الازرار التى تقدمها.



شكل (٩-٤): ان ال MessageDlg dialog box يمكنك من اختيار ايقونة، مثل دائرة المعلومات الموضحة هنا، للعرض

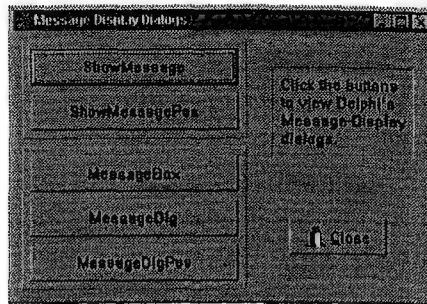
• ان ال MessageDlgPos procedure هو نفسه ال MessageDlg بخلاف انه يقدم إحداثيات x- و y- لوضع نافذة ال dialog فى أى مكان على شاشة الحاسب المكتبى للـ Windows ، كما هو موضح فى شكل (٩-٥).



شكل (٩-٥): يمكنك وضع ال MessageDlgPos dialog فى أى مكان حاسب المكتبى للـ Windows

الـ MessageDlg والـ MessageDlgPos عبارة عن procedures تعرض رسائل. وهى مناسبة فقط لعرض الرسائل الثابتة. ال MessageBox، والـ MessageDlg، والـ MessageDlgPos تعتبر functions، وهى تدخل قيمة تشير الى أى من الازرار تم ضغطه لإغلاق نافذة ال dialog استخدم واحدة من هذه ال functions لحث المستخدمين على إجابة اسئلة أو تأكيد اختيار- مثلاً، " هل انت متأكد تماماً من انك تريد مسح محرك قرصك الصلب؟ " .

يوضح تطبيق ال MsgDlg على القرص المدمج فى دليل ال Source\MsgDlg الانواع الخمسة من ال dialogs لعرض الرسالة. ان جميع الطرق الا واحدة يسهل استخدامها. أما ال MessageBox فتطلب استخدام تقنيات string لم يتم تقديمها بعد. سوف اشرح هذا عقب القائمة. يوضح شكل (٩-٦) عرض البرنامج. تظهر القائمة (٩-١) ال source code .



شكل (٩-٦) : قم بتشغيل الـ `MsgDlg` لتجربة  
الأنواع الخمسة للـ `dialogs` لعرض الرسالة

القائمة (٩-١) : `Msgdlg\Main.pas`

```
unit Main;
```

```
interface
```

```
uses
```

```
  SysUtils, Windows, Messages, Classes, Graphics,  
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls, Buttons;
```

```
type
```

```
  TMainForm = class(TForm)
```

```
    Button1: TButton;
```

```
    Button2: TButton;
```

```
    Button3: TButton;
```

```
    Button4: TButton;
```

```
    Button5: TButton;
```

```
    BitBtn1: TBitBtn;
```

```
    Label1: TLabel;
```

```
    Panel1: TPanel;
```

```
    Bevel1: TBevel;
```

```
    Panel2: TPanel;
```

```
    procedure Button1Click(Sender: TObject);
```

```
    procedure Button2Click(Sender: TObject);
```

```

procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    private
    { Private declarations }
    public
    { Public declarations }
    end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.Button1Click(Sender: TObject);
begin
    ShowMessage('Procedure: ShowMessage');
end;

procedure TMainForm.Button2Click(Sender: TObject);
begin
    ShowMessagePos('Procedure: ShowMessagePos', 10, 20);
end;

procedure TMainForm.Button3Click(Sender: TObject);
var
    TheText, TheCaption: String;
begin
    TheText := 'Function: MessageBox';
    TheCaption := 'MessageBox Demonstration';
    if Application.MessageBox(PChar(TheText), PChar
(TheCaption),

```

## Single-Line Strings مع العمل : الباب التاسع

```
MB_DEFBUTTON1 + MB_ICONEXCLAMATION + MB_OKCANCEL)= IDOK
    then ShowMessage('You selected OK')
    else ShowMessage('You selected Cancel');
end;
```

```
procedure TMainForm.Button4Click(Sender: TObject);
var
    W: Word;
    S: String;
begin
    W := MessageDlg('Function: MessageDlg',
        mtInformation, [mbYes, mbNo, mbIgnore], 0);
    case W of
        mrYes:  S := 'Yes';
        mrNo:   S := 'No';
        mrIgnore: S := 'Ignore';
    end;
    ShowMessage('You selected ' + S);
end;
```

```
procedure TMainForm.Button5Click(Sender: TObject);
var
    W: Word;
    S: String;
    X, Y: Integer;
begin
    X := 50; Y := 75;
    W := MessageDlgPos('Function: MessageDlgPos',
        mtWarning, mbAbortRetryIgnore, 0, X, Y);
    case W of
        mrAbort: S := 'Abort';
        mrRetry: S := 'Retry';
        mrIgnore: S := 'Ignore';
    end;
```

```
end;
  ShowMessage('You selected ' + S);
end;

end.
```

ان الـ ShowMessage سهلة الاستخدام . فقط قم بتمرير ايه string أو string variable لعرض رسالة تذكر ان Delphi يسمح لك بربط الـ strings بعامل الزائد- وهذا ما يجعل الـ ShowMessage سهلة لعرض كل انواع القيم ولإزالة اخطاء المتغيرات . على سبيل المثال ، لعرض قيمة متغير الـ Integer Count ، استخدم عبارة مثل هذه :

```
ShowMessage('Count = ' + IntToStr(Count));
```

والـ ShowMessagePos هو نفسه الـ ShowMessage ولكن يضيف قيم احداثيات الـ x- والـ y- ذات علاقة بالشاشة . لعرض الرسالة السابقة فى الركن الایسر العلوى لحاسب الـ Windows المكتبى ، استخدم هذا العبارة :

```
ShowMessagePos('Count = ' + IntToStr(Count), 0, 0);
```

يدخل الـ MessageDlg والـ MessageDlgPos قيمة تشير الى الزر الذى تم ضغطه لإغلاق نافذة الـ dialog . لانك تقرر أى الازرار يعرضه الـ dialog ، فعليك ان تبحث عن هذه الازرار فقط- إنك لا تحتاج الى كتابة كود للاستجابة لكل قيمة مدخلة محتملة . ان عبارة case عادة تكون افضل لإنشاء أولاً ، حدد متغير الـ Word وهو W وعين له MessageDlg أو MessageDlgPos بعبارة مثل هذه :

```
W := MessageDlg('Function: MessageDlg',
  mtInformation, [mbYes, mbNo, mbIgnore], 0);
```

ثم استخدم عبارة case للقيام بعمل مناسب يعتمد على الزر الذى اختاره المستخدم :

```
case W of
  mrYes:  { action for Yes  };
  mrNo:   { action for No   };
  mrIgnore: { action for Ignore };
end;
```

## الباب التاسع : العمل مع Single-Line Strings

ان ال MessageDlgPos هي تقسها ال MessageDlg ولكن تضيف احداثيات ال x- و ال y-، لوضع ال dialog على حاسب ال Windows المكتبي . ان الاحداثيات لها علاقة بالشاشة ، ولكن يمكنك تحويلها الى قيم مرتبطة بال client لعرض dialogs داخل مساحة نافذة البرنامج . وهذا افضل لواجهة التطبيق لان المستخدمين يميلون الى البحث عن الرسائل فى مواضع محددة مرتبطة بنافذة التطبيق .

ان ال MessageBox يعتبر اكثر تعقيداً فى الاستخدام . إنك تحتاج ان تمرر له null-terminated strings زائد مجموعة منطقية من الثوابت مثل ال MB\_OKCANCEL لاختيار خيارات متنوعة :

```
function MessageBox(Text, Caption: PChar;  
Flags: Longint): Integer;
```

الأثنان من parameters من نوع ال PChar ، والتي تمثل pointer الى رمز فى Delphi . لان ال strings الطويلة تكون null terminated ، يمكنك وضعهم فى ال pointers ال PChar للتمرير الى ال Windows functions مثل هذه . على سبيل المثال ، يقرر ال Button3Click declare إثنين من متغيرات ال string :

```
var
```

```
TheText, TheCaption: String;
```

يقوم البرنامج بتعيين strings حرفية الى هذه المتغيرات باستخدام عبارتين :

```
TheText := 'Function: MessageBox';  
TheCaption := 'MessageBox Demonstration';
```

قم بتمرير اثنان من ال string ، type-cast الى ال PChar pointers ، من ثم الى ال MessageBox function . يقوم البرنامج بهذا باستخدام عبارة if لاختيار نتيجة ال function ، والتي تشير الى الزر الذى تم ضغطه لإغلاق نافذة dialog box توضح عبارة ال if كيف تضع strings فى ال PChar pointers :

```
If Application.MessageBox(PChar(TheText), PChar  
(TheCaption),  
MB_DEFBUTTON1 + MB_ICONEXCLAMATION +  
MB_OKCANCEL) = IDOK  
then ShowMessage('You selected OK')  
else ShowMessage('You selected Cancel');
```



ان التعبيرات PChar (Text) و PChar (Caption) تمرر متغيرات string الطويل الى الـ Windows function ، والتي تتوقع أن تتلقى null-terminated strings فى هذه المواضع . تقوم الـ MessageBox function بإدخال قيمة الزر الذى ضغطه المستخدم . اذا كانت مساوية لـ IDOK ، اذن يتم إظهار الرسالة الأولى باستخدام الـ ShowMessage . اذا لم تكن كذلك ، يكون المستخدم قد ضغط Cancel ، ويتم عرض الرسالة الثانية .

وتوضح الـ Button3Click فى القائمة كيفية استخدام الـ MessageBox ، ولكننى اقترح عليك أن تستخدم بدلاً منه الـ MessageDlgPos والـ MessageDlg parameters . يقوم Delphi بتعريف الـ MessageDlg parameters الخاصة به كما يلى :

```
function MessageDlg(const Msg: string; AType: TMsgDlgType;
  AButtons: TMsgDlgButtons; HelpCtx: Longint): Word;
```

• **Msg**: تمرر string ثابت أو متغير حرفى للعرض فى dialog box .

• **AType**: حدد هذا الـ parameter بثابت مثل الـ mtWarning أو الـ mtError لإختيار الأيقونة المناسبة و dialog caption .

• **AButtons**: حدد هذا الـ parameter بمجموعة الأزرار التى تريد عرضها فى الـ dialog . ضع المجموعة بين قوسين . على سبيل المثال ، مرر التعبير [mbYes, mbNo, mbCancel] لعرض أزرار Yes أو No أو Cancel . يضيف Delphi تلقائياً الـ glyphs المناسبة لكل زر (إنها الـ BitBtn component object) . وبالعكس ، بدلاً من تحديد مجموعة الأزرار الخاصة بك ، قم بتمرير مجموعة سابقة التحديد ، مثل الـ mbOkCancel ، mbYesNoCancel أو myAbortRetryIgnore . فهذه هى بالفعل مجموعات Pascal- لا تحيطها بالاقراص .

• **HelpCtx**: حدد هذا الـ parameter بصفر الا اذا كنت تحدد نص مساعدة message box .

## الباب التاسع : العمل مع Single-Line Strings

انظر ال MessageDlg لمزيداً من المعلومات عن القيم الممكن ان تمررها لهذه ال function وللا MessageDlgPos .

**على القرص المدمج:** توضح القائمة (٩-٢) كيفية استخدام الاحداثيات المتعلقة بالشاشة لوضع عناصر في النافذة . حدد متغير ال TPoint بـ P ، حدد اعضاء ال X وال Y التابعين له بـ (0) ، واستدع ال ClientToScreen ، والذي يدخل سجل TPoint بهذه القيم تم تحويله الى إحداثيات شاشة مساوية . يمكنك عندئذ استدعاء MessageDlgPos كما هو موضح (أو استدع ShowMessagePos ) لوضع message box في الركن الایسر العلوی أو فی موضع آخر من ال client area بنافذة التطبيق . يوجد نص القائمة على القرص المدمج في دليل Strings في ملف ال Position.pas .



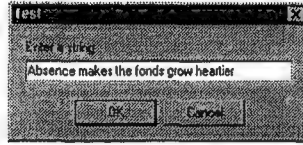
القائمة (٩-٢): استخدم هذا ال code لوضع message dialog داخل نافذة التطبيق.

```
var
  W: Word; { Result of MessageDlgPos }
  P: TPoint; { Coordinate X, Y record }
begin
  P.X := 0; { Assign client X value }
  P.Y := 0; { Assign client Y value }
  P := ClientToScreen(P); { Convert P to screen coordinates }
  W := MessageDlgPos('Function: MessageDlg',
    mtInformation, [mbYes, mbNo, mbIgnore], 0,
    P.X, P.Y); { Pass converted X, Y to function }
end;
```

إذا قمت بتمرير ال X وال Y مساوية لـ 1- لل ShowMessagePos أو ال MessageDlgPos ، تعرض ال functions نوافذ ال dialog الخاصة بها في الاماكن الافتراضية التي اختارها ال Windows .

## البحث على إدخال النص:

على القرص المدمج: ان البرمجة عادة هي عملية اخذ وعطاء . ان message dialogs السابقة تعطى المستخدمين معلومات . لكى تأخذ إدخالاً من المستخدمين ، استدع الـ InputBox أو الـ InputQuery والتي تعرض الـ dialog الموضح فى شكل (٧-٩) . الـ InputBox ترجع بـ string؛ الـ InputQuery ترجع بقيمة Boolean بـ True أو False ، ولكن فيما عدا هذا تعتبر كل الـ functions متساويتين . توضح القائمة (٣-٩) كيفية استخدام الـ InputBox والـ InputQuery . انسخ هذا الـ procedures الى وحدة برنامج مثل OnClick لإثنين من الـ Button objects على الـ form ، وقم بتشغيل البرنامج لتجربته مع الـ InputBox functions والـ InputQuery . هذا ليس برنامجاً كاملاً- لتستخدم الـ code ، أضف اثنان من الـ Button objects على الـ form وانسخ الـ procedures فى OnClick لكل Button . يوجد نص القائمة على القرص المدمج فى دليل Strings فى ملف Input.pas .



شكل (٧-٩): استدع الـ InputBox أو الـ InputQuery لبحث المستخدمين على إدخال single-line text strings تعرض الـ functions الـ dialog الموضح هنا

### القائمة (٣-٩): استخدام الـ InputBox والـ InputQuery

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S: String;
begin
  S := InputBox('Test', 'Enter a string', '');
  if Length(S) > 0 then
    ShowMessage('You entered: ' + S);
end;
```

```

procedure TForm1.Button2Click(Sender: TObject);
var
  S: String;
begin
  S:= ' ' ;
  if InputQuery('Test', 'Type QUIT to end', S) then
    if Uppercase(S) = 'QUIT' then
      Close;
end;

```

تأخذ الـ `InputBox` ثلاث `string` arguments وهم الـ `caption`، `prompt`، وقيمة `string`، والتي يمكن ان تكون متغيرة أو ثابتة. تمرر القائمة `string null` للـ `InputBox` للـ `string` الافتراضى، ويتم إدخاله اذا لم يدخل المستخدمون اية تغييرات. يعرض الـ `InputBox` الـ `string` الافتراضى وقد تم إبرازه فى الـ `dialog` وتقوم الـ `function` أيضاً بإدخال الـ `string` الافتراضى اذا اضغط المستخدم زر الـ `control`. اذا اضغط المستخدم زر الـ `OK`، الـ `InputBox` يرجع النص الذى تم إدخاله فى الـ `Edit control` للـ `dialog`. على سبيل المثال، استخدم الـ `InputBox` للبحث على إدخال اسم ملف:

```

S := InputBox('Test', 'Enter filename', 'Readme.Txt');
ShowMessage('You entered: ' + S);

```

**على القرص المدمج:** ان الـ `InputQuery` هو نفسه الـ `InputBox`، ولكنه يدخل `True` اذا ضغط المستخدم زر الـ `OK`، أو `False` اذا ضغط زر الـ `Cancel`. يضع الـ `InputQuery` إدخال المستخدم فى الـ `string` الافتراضى، والذي يجب ان يكون متغير. تعتبر الـ `functions` نموذجية فى حث المستخدمين على إدخال الـ `strings` مثل اسماء الملفات الافتراضية للمسارات المعروفة. على سبيل المثال، أضف `Button` و `Memo` على الـ `form`، ثم استخدم البرمجة الموجودة فى القائمة (٩-٤) للبحث على إدخال اسم ملف، وهو الافتراضى الـ `Readme.txt` الخاص بهذا للكتاب فى دليل الـ `Source`. يتم توضيح نص الملف المختار فى `Memo object`. يوجد نص القائمة فى القرص المدمج فى دليل `Strings` فى ملف `Query.pas`.



القائمة (٩-٤): كيفية تحميل ملف معين

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Filename: String;
begin
  Filename := 'Delphi 4 Bible\Source\Readme.Txt';
  if InputQuery('Test', 'File?', Filename) then
    Memo1.Lines.LoadFromFile(Filename);
end;
```

## : Single-Line Text Components

يقدم Delphi ثلاث components لإخراج وادخال نص احادى السطر، وهى: Label، Edit، و MaskEdit. أنها components سهلة الاستخدام نسبياً، ولكن الفصول التالية توضح التقنيات الغامضة التى يجب ان تعرفها.

### : Labels

ان ال Labels ببساطة text objects يمكنك إضافتها على forms و components أحادية اخرى، مثل ال Panel. كما اكتشفت من خلال امثلة هذا الكتاب حتى الآن، ان لل Labels خصائص متعددة. يمكنك تحديد ال Font الخاص بال Labels لتغيير نمط نصه، يمكنك عرض wrap-around بتحديد ال WordWrap بـ True، يمكنك تغيير لون ال Labels من خلال خاصية ال Color.

**فكرة:** تختار ال Color لون خلفية ال Labels. استخدم خاصية ال Font لاختيار لون نص واجهة ال Labels.

فى اغلب الحالات، تعتبر ال Labels غير متفاعلة- فهى ببساطة تضع label component آخر، أو تعطى بعض معلومات على الشاشة. وتعتبر واحدة من الاستخدامات الغير واضحة لل Label هى التعامل معه كـ hot key accelerator لـ control آخر لا يكون فى العادة متوافق مع اختيارك لأخذ عناصر ال keyboard. إتبع هذه الخطوات لتجرب هذه التقنية المفيدة:

## الباب التاسع : العمل مع Single-Line Strings

- ١- أضف ثلاثة Edit objects و ثلاثة Label objects على form . ضع label الى اليسار مباشرة من كل ال Edit control .
  - ٢- غير ال Caption الخاص بال Label1 الى &One . تحدد علامة ال & حرفاً تالياً (وهو O في هذه الحالة) ك accelerator hot key .
  - ٣- اختر خاصية ال FocusControl لل Label1 ، اضغط السهم المشير الى أسفل ، واختر Edit1 . عندما يتلقى ال Label1 input focus ، فإنه يمرر ال focus الى ال control المشار إليه .
  - ٤- أعد الخطوات ٢ , ٣ لل Label الآخرين . حدد ال Caption الخاص بال Label2 بـ &Two ، والخاص بال Label3 بـ T&hree . عين ال FocusControl الخاص بال Label2 لل Edit2 ، والخاص بال Label3 لل Edit3 .
  - ٥- قم بتشغيل البرنامج واضغط Alt+O ، Alt+T ، و Alt+H لتحويل ال focus الى ال labeled Edit boxes .
- لوضع ال Label أفقياً مع ال Edit أو أى control آخر متعلق به ، اختر أولاً ال Edit control ، اضغط واستمر في ضغط مفتاح ال Shift ، واضغط ال Label المقابل . اختر أمر ال EditAlign... الخاص بـ Delphi ، اختر Vertical Centers ، واضغط Enter أو OK .

### ١. Edit boxes :

- للتعامل مع single-line input ، لا يوجد ما يضاهي ال Edit component الخاص بـ Delphi . فيما يلي بعض الأفكار عن الخصائص المختارة ال Edit component .
- \* حدد ال AutoSelect بـ True لإبراز محتويات ال Edit عندما يتلقى ال input focus .
  - \* حدد ال CharCase بـ ecNormal للإدخال العادى حدده بـ ecUpperCase أو ecLowerCase لقصر الإدخال على الأحرف الكبيرة أو الحروف الصغيرة .

\* حدد ال HideSelection بـ False اذا كنت تريد نص تم إبرازه أن يبقى مختاراً عندما يحول المستخدمون ال focus الى control آخر . حدد ال HideSelection بـ True العملية العادية- النص المختار غير محدد عندما ينتقل ال focus من ال Edit control .

\* حدد ال MaxLength بأكبر عدد من الرموز تريد أن يدخلها المستخدمون . بالرغم من أنه بإمكانك أن تحدد هذه الخاصية بصفر للحصول على طول للدخال غير محدد، فمن الناحية العملية، إنك تستخدم قيم صغيرة نسبياً . إن حجم النص الذى يمكنك إدخاله فى Edit boxes هو فى الحقيقة غير محدد . فى النسخ الأولى للـ Delphi ذات ال ١٦ بت ، كان النص محدد بـ ٢٥٥ رمزاً . ولقد تم إزالة هذا الحد فى نسخ ال ٣٢ بت من Delphi ؛ ولكن ال Windows 95 و 98 (ولكن ليس ال NT) يقصرا النص على 64K كحد أقصى .

\* إن ال Edit component ليس له خاصية Caption . استخدم ال Label ، كما هو موضح فى الفصل السابق ، لإعطاء ال Edit objects ال captions و hot-key accelerators لعرض ال object ، أضف رموزاً فى خاصية ال Text ، أو عين string فى وقت التشغيل .

\* لعرض رموز DOS على نص الشاشة ، حدد ال OEMConvert بـ True ، وغير ال Font بـ TrueType MS LineDraw . حدد ال OEMConvert بـ False اذا لم تكن تريد رموز ANSI محولة (هذا هو التحديد الطبيعى للـ Windows) . لتأييد رموز ال ASCII الممتدة فى اسم ملف ال DOS ، يجب أن يكون ال OEMConvert محدد بـ True .

وتوفر ال Edit components أيضاً events عديدة . وبعض هذه ال events المفيدة هى :

● **OnChange**: يتم استدعاه عندما تتغير محتويات ال control بأية طريقة . اذا قمت بتشغيل هذا ال event ، فإنتبه الى أنه يتم استدعاه لكل رمز يتم إدخاله فى ال control . للتعامل مع الحروف حرفاً وحرفاً والمُدخلة فى ال Edit control ، فيجب أن تستخدم ال OnKeyPress event .

## الباب التاسع : العمل مع Single-Line Strings

● **OnClick:** يتم استدعائه عندما يضغط المستخدم الفأرة مرة واحدة في ال Edit control . لاحظ أن ال event handler هذا لا يتم استدعائه عندما يتلقى ال Edit control ال focus بطرق أخرى - عندما يضغط المستخدم ال Tab مثلاً .

● **OnDbClick:** يتم استدعائه عندما يضغط المستخدم الفأرة مرتين في ال Edit control . والإستجابة العادية لهذا ال events تنسخ خاصية ال Text ل Edit control الى object آخر مثل ال ListBox .

● **OnEnter:** يتم استدعائه عندما يتلقى ال control ال input focus - أى عندما يختار المستخدم ال control ويضغطه بالفأرة .

● **OnExit:** يتم استدعائه عندما يفقد ال control ال input focus - أى عندما يحول المستخدم ال focus الى control آخر .

● **OnKeyPress:** يتم استدعائه مع كل ضغطه مفتاح . بشكل عام هذا ال event هو الذى يصبح استخدامه لتتقية إدخال لوحة المفاتيح على سبيل المثال ، لمنع المستخدمين من ضغط مفاتيح معينة ، اذا كان ال Key parameter الخاص بال event handler يساوى قيمة مستهدفة ، حدد ال Key ب #0 حتى تبطله .

### استخدام ال OnExit وال OnEnter

من الناحية العملية ، تعتبر ال OnEnter وال OnExit مفيدة فى تصميم واجهات تطبيق جذابة . على سبيل المثال ، يمكنك استخدام هذه ال events لتلوين ال Edit control focus ، والذى يمكن أن يساعد فى لفت إنتباه المستخدم الى المجال الحالى فى نافذة ذى مساحات إدخال متعددة . عرف متغير OldColor من نوع ال TColor فى القطاع private فى ال form class ، كما يلى :

```
TForm1 = class(TForm)
```

```
    Edit1: TEdit;
```

```
private
```

```
    OldColor: TColor;
```

```
...
```

```
public
```

```
...
```

```
end;
```



بعد ذلك ، قم بإنشاء OnEnter للـ Edit. عين الـ Color الخاص بالـ object لتغيير الـ OldColor ثم اختر اللون الذى تريده لخلفية الـ control. مع هذا الـ code، عندما يقوم المستخدمون بتحويل الـ focus الى الـ control، كيف تتغير خلفيته الى اللون الأصفر:

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
    OldColor := Edit1.Color;
    Edit1.Color := clYellow;
end;
```

وكذلك ، قم بالـ event handler للـ OnExit الخاص بالـ Edit object. عين قيمة الـ OldColor لخاصية الـ Color للـ object. عندما يقوم المستخدمون بتحويل الـ focus عن الـ control، يستعيد البرنامج خلفية الى اللون الذى تم حفظه.

```
procedure TForm1.Edit1Exit(Sender: TObject);
begin
    Edit1.Color := OldColor;
end;
```

### :Telephonic mnemonics

**على القرص المدمج:** إن تطبيق الـ TelName، الموجود على القرص المدمج فى دليل الـ Source\TelName، يظهر كيفية استخدام الـ Edit objects. وهذا البرنامج يستخدم Delphi لرسم وجه جديد على بعض الـ codes التى كتبتها للعمود Dr. Dobb الخاص بى، وهو "Algorithm Alley" منذ عدة سنوات. ولقد حل البرنامج واحدة من مشاكل الدائمة-سرعة التذكر للصيغة الهاتفية الموجودة فى اعلانات مثل "اتصلوا بنا الآن، فقط اتصل بـ ٨٠٠-١-اتصل-بنا-الآن". وهذا يستغرق منى وقتاً طويلاً لأجد الأرقام المقابلة فى قرص الهاتف، لذا كتبت برنامجاً لتحويل الصيغة الى أرقاماً. ولقد كتبت أيضاً الـ code العكسية التى تعرض كل البدائل الممكنة لأى رقم. فى منطقتى مثلاً، الإستبدال العام ٢٩٣ يمكن أن ينهى كلمات مثل AWE، AXE، و AYE، مؤدياً الى صيغ هاتفية ممتعة.



## الباب التاسع : العمل مع Single-Line Strings

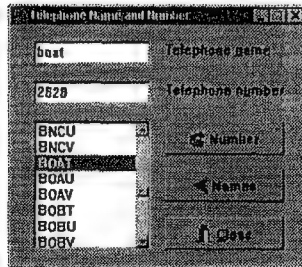
لاستخدام الـ TelName، أدخل صيغة هاتفية في Edit control لاسم Telephone. اضغط زر الـ Number لتحويل الاسم الى ما يوازيه بالأرقام. أو، أدخل رقماً في الـ Edit control لاسم الـ Telephone، واضغط Names لإدخال كل بدائل الصيغ الممكنة في list box النافذة. ملحوظة: على قرص الهاتف، الأرقام واحد وصفر ليس لها حروف مقابلة، والحروف Q و Z ليس لها أرقام مقابلة.

إن أرقام الهاتف الطويلة في الـ TelName تنتج مداخل ListBox متعددة، والتي قد تؤدي الى بالـ Windows الى حالة run out of resources. إن عدد المدخلات يساوي  $3n$  و  $n$  هي عدد الأرقام. فالأربع أرقام تنتج ٨١ مدخلاً. والستة أرقام تنتج ٧٢٩ مدخلاً. للحصول على أفضل النتائج، اكتب ثلاثة أو أربعة أرقام كحد أقصى في الـ Edit لرقم الـ Telephone قبل ضغط الـ Names. على سبيل المثال، لترى ما عناصر رقم هاتفك، بدلاً من إدخال كل الأرقام، أدخل code المنطقة، والبديل له رقم بصورة منفصلة.

**ملحوظة:** بسبب تحكم الذاكرة الفائق، فهذا لا يشكل عبءاً للـ Windows 95، 98، وخاصة الـ NT كما كان في الـ Windows 3.1.



يوضح شكل (٨-٩) عرض الـ TelName. لأفضل النتائج، إقصر العدد على ثلاثة أو أربعة أرقام قبل ضغط Names. توضح القائمة (٩-٥) source code للبرنامج. للحصول على شرح عن كيفية عمل الـ algorithm، وعن البدائل بشكل عام، أنظر مقالين "Algorithm Alley: Telephonic Mnemonics"، في صحيفة Dr. Dobb's Journal بتاريخ يونية ١٩٩٣.



شكل (٨-٩): يحول الـ TelName الصيغ الهاتفية الى أرقام هاتفية مساوية، ويمكنه أن يظهر كل البدائل الأبجدية الممكنة لأي رقم

## القائمة (٩-٥) TelName\Main.pas

```

unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
  Controls,
  Forms, Dialogs, StdCtrls, Buttons, Mask;

type
  TMainForm = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    ListBox1: TListBox;
    CloseBitBtn: TBitBtn;
    NamesBitBtn: TBitBtn;
    NumberBitBtn: TBitBtn;
    NameEdit: TEdit;
    NumbEdit: TEdit;
    procedure FormCreate(Sender: TObject);
    procedure NumberBitBtnClick(Sender: TObject);
    procedure NamesBitBtnClick(Sender: TObject);
    procedure NumbEditKeyPress(Sender: TObject; var Key:
    Char);
    procedure NameEditKeyPress(Sender: TObject; var Key: Char);
  private
    { Private declarations }
    function TelNameToNum(TelName: String): String;
    procedure ListNames(TelNum: String);
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

```

implementation

{ \$R \*.DFM }

var

{ Array of telephone dialog letters }

TelDial: array[0 .. 9] of String;

{ Return telephone digit that corresponds to C }

function DigitToLetter(C: Char): Char;

var

I, J: Integer;

begin

C := Upcase(C);

for I := 0 to 9 do

for J := 1 to 3 do

if (C = TelDial[I][J]) then

begin

Result := Chr(I + Ord('0'));

Exit;

end;

Result := C; { Default }

end;

{ Return ordinal value of digit character C }

function ValueOfChar(C: Char): Integer;

begin

Result := Ord(C) - Ord('0');

end;

{ Return number for a telephone alphabetic name }

function TMainForm.TelNameToNum(TelName: String): String;

var

I: Integer;

begin

```

Result := "";
for I := 1 to Length(TelName) do
    Result := Result + DigitToLetter(TelName[I]);
end;

procedure TMainForm.ListNames(TelNum: String);
var
    S: String; { Temporary string }

    { Find N alphabetic permutations of digits in TelNum }
    procedure Permute(N: Integer);
    var
        I, Digit: Integer;
    begin
        Digit := ValueOfChar(TelNum[N]);
        for I := 1 to 3 do
            begin
                S[N] := TelDial[Digit][I]; { Insert letter }
                if (N = Length(TelNum)) then
                    ListBox1.Items.Add(S) { Add string to ListBox }
                else
                    Permute(N + 1); { Call Permute recursively }
                end;
            end; { Permute }
        end;

begin
    if Length(TelNum) > 0 then
        begin
            S := TelNum; { Assign to temporary string }
            Permute(1); { Start permutations }
        end;
end;

{ Initialize global variables }
procedure TMainForm.FormCreate(Sender: TObject);
begin

```

## الباب التاسع : العمل مع Single-Line Strings

```
TelDial[0] := ' '; TelDial[1] := ' ';
TelDial[2] := 'ABC'; TelDial[3] := 'DEF';
TelDial[4] := 'GHI'; TelDial[5] := 'JKL';
TelDial[6] := 'MNO'; TelDial[7] := 'PRS';
TelDial[8] := 'TUV'; TelDial[9] := 'WXY';
end;

{ Do Number button click }
procedure TMainForm.NumberBitBtnClick(Sender: TObject);
begin
    NumbEdit.Text := TelNameToNum(NameEdit.Text);
    NumbEdit.SetFocus;
end;

{ Do Names button click }
procedure TMainForm.NamesBitBtnClick(Sender: TObject);
begin
    ListBox1.Clear;
    ListNames(NumbEdit.Text);
    ListBox1.SetFocus;
end;

{ Click Names button for Enter key in Number Edit object }
procedure TMainForm.NumbEditKeyPress(Sender: TObject;
var Key: Char);
begin
    if Key = #13 then
    begin
        NamesBitBtn.Click;
        Key := #0;
    end;
end;

{ Click Number button for Enter key in Name Edit object }
procedure TMainForm.NameEditKeyPress(Sender: TObject;
var Key: Char);
```

```

begin
  if Key = #13 then
    begin
      NumberBitBtn.Click;
      Key := #0;
    end;
  end;

end.

```

يوضح الـ TelName بعض واجهات التطبيق المفيدة وتقنيات string-handling. وهناك إثنين من الـ labels توفران accelerator hot keys لـ Edit controls، باستخدام الـ methods الموضحة سابقاً. على سبيل المثال، تضع الـ Telephone name label قبل الـ \$ فى علامة الـ \$، وخاصية الـ Control فى الـ Label محددة بـ NameEdit. مع قيم هذه الخصائص، فإن ضغط Alt+N يختار الـ labeled Edit object.

وتعرض أزرار الـ Names والـ Number الخاصة بالبرنامج كيفية تحويل الـ focus الى control آخر بعد اختيار الزر على سبيل المثال، عندما تضغط زر الـ Number يحول البرنامج الصيغة الهاتفية فى الـ Edit لاسم الـ Telephone الى رقم. قد يريد المستخدمون تعديل هذا الرقم للذكر حروف اخرى ممكنة، لذا يقوم الـ OnClick الخاص بزر الـ Number بتحويل الـ focus بعيداً عن نفسه الى الـ NameEdit. تتعامل العبارتان التاليتان مع اختيار زر الـ Number:

```

NumbEdit.Text := TelNameToNum(NameEdit.Text);
NumbEdit.SetFocus;

```

تحول العبارة الأولى النص الخاص بـ NameEdit object، boat-Edit مثلاً- الى رقم الهاتف المساوى (2628). وتستدعى العبارة الثانية الـ SetFocus method الخاص بالـ NameEdit لنقل الـ focus الى الـ Edit object التالى. ان ضغط زر الـ Names يؤدي مهمة مشابهة باستخدام هذه العبارات الثلاث فى OnClick:

```
ListBox1.Clear;
```

## الباب التاسع : العمل مع Single-Line Strings

```
ListNames(NumbEdit.Text);
```

```
ListBox1.SetFocus;
```

أولاً، يسمح الـ procedure المدخلات الحالية للـ ListBox. ثم يستدعى بعد ذلك الـ ListNames الذى يبدل رقم هاتف بكل الصيغ الابدجية الممكنة. وتحول العبارة الاخيرة الـ input focus الى الـ ListBox حتى يمكن للمستخدمين ضغط مفاتيح الاسهم لرؤية النتائج الواردة.

ان توفير عمليات لوحة مفاتيح فعالة للبرامج الجرافيكية يعتبر سحراً اسود لا يمارس بشكل جيد عادة. بالإضافة الى السماح بتحكم لوحة المفاتيح على الأوامر، تستخدم واجهة تطبيق المستخدم بشكل جيد methods مثل الـ SetFocus لتقليل التعامل مع المفاتيح. بالرغم من انه من المأمون ان نقول ان كل نظم الـ Windows لديها وسيلة ادخال بالفأرة، فالتطبيق التام يجب ان يكون من الممكن تنفيذ عملياته من لوحة المفاتيح.

الـ TelName يتخذ الـ procedures آخرين لتحسين التعامل مع لوحة المفاتيح. عندما تضغط Enter فى الـ input focus عادة بالتحويل الى الـ control التالى فى ترتيب الـ Tab. هذا يوفر فرصة ممتازة لأداء ايه عمليات لازمة على محتويات الـ Edit control، مما يزيل ضغطه زر واحدة. قم بتشغيل الـ TelName وإدخل الـ boat فى الـ Edit control لاسم الـ Telephone. عندما تضغط Enter، يحول البرنامج الـ focus الى منطقة الـ Edit برقم الـ Telephone، ويحول الـ boat الى الرقم المساوى لها. اضغط Edit مرة اخرى للتحويل الى الـ ListBox وإدخل كل البدائل الممكنة لهذا الرقم. عندما تجرب هذا، سيدو والتأثير واضحاً، ولكن التوصل الى جهاز لوحة مفاتيح بصورة سهلة يتطلب عادة مزيداً من التفكير والاختيار.

لإعادة برمجة مفتاح الـ Edit لتنفيذ هذه المهام المتفاعلة، يستخدم الـ TelName إثنين من الـ OnKeyPress event handlers للـ NameEdit objects والـ NumbEdit. فى الـ NumbEditKeyPress، مثلاً، يبحث البرنامج عن carriage return (يمثل التعبير #13 رمز ASCII بقيمة ١٣). لمعالجة المعلومة فى نافذة الـ Edit، يحاكي البرنامج ضغطه زر بعبارة:



NamesBitBtn.Click;

بعد ذلك، يلقي البرنامج بال carriage return لان الزر يحول بالفعل ال focus الى نافذة ال Edit التالية، ونحن لا نريد ان يتلقى ال control الحالى هذا الرمز. للتخلص من ضغطة مفتاح ال Edit، حدد ال Key بـ ASCII لإلغاء هذا التعيين:

Key := #0;

ال procedure ListNames يستدعى ال Permute procedure المتداخل لايجاد كل المجموعات الابدجية لرقم تم تمريره كـ TelNum parameter.

### :Masked Edit boxes

يوفر ال MaskEdit component هيئة ال Edit والذي يحدد للمستخدمين نوعية الرموز المطلوبة ادخالها فى مكان محدد. وهناك secondary string، يدعى ال EditMask، يحدد قواعد الادخال. على سبيل المثال، يمكن ان يحدد ال mask ال format الخاصة برقم هاتف والذي يمدد المستخدمين ان يدخلوا رقم هاتف مكون من سبعة ارقام. اتبع هذه الخطوات لتجربة ال MaskEdit:

١- أضف ال MaskEdit من ال Additional VCL palette على ال form. وكذلك أضف ال Edit object و Button.

٢- يقدم ال MaskEdit خاصية، وهى ال EditMask، تحدد تنسيق الادخال الخاص بال control. يمكنك تعيين قناعاً فى وقت التصميم بتعديل هذه الخاصية. ولفعل نفس الشئ فى وقت التشغيل، اضغط ال Button1 مرتين وادخل هذه العبارة فى ال OnClick الخاص بها:

MaskEdit1.EditMask := Edit1.Text;

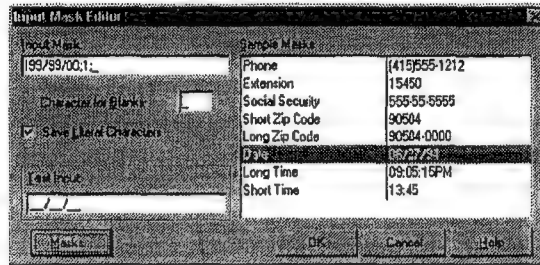
MaskEdit1.SetFocus;

٣- قم بتشغيل البرنامج وإدخل 1;9999-999 فى ال Edit1 object. ادخل كل علامات الترقيم كما هو موضح (لاحظ underscore الأخيرة). اضغط الزر لنسخ هذا النص الى خاصية ال EditMask لـ MaskEdit. يحدد ال mask رقم هاتف من سبعة ارقام (سوف اشرح المزيد عن تنسيق ال mask فيما بعد).

## الباب التاسع : العمل مع Single-Line Strings

٤ - إدخال نصاً في الـ MaskEdit لتجربة الـ mask .

يمكنك أيضاً اختيار الـ mask بضغط الزر البيضاوى لخاصية الـ EditMask لـ MaskEdit. هذا يفتح الـ Input Mask Editor dialog الخاص بـ Delphi، الموضح في شكل (٩-٩) اختر الـ mask، وإدخل strings اختبارية في الـ Test Input control. قم بتحميل ملفات الـ dem. (Delphi Edit Mask) بضغط زر الـ Masks. يقدم Delphi ملفات عديدة ذات ارقام هاتف، تواريخ، اوقات، و formatting masks اخرى بعدة لغات. توضح القائمة (٩-٦) نصاً من الـ Denmark.dem، الموجود في الدليل الفرعى Bin الخاص بـ Delphi.



شكل (٩-٩): الـ Input Mask Editor dialog الخاص بـ Delphi يقدم mask لخاصية الـ EditMask التابعة لـ MaskEdit

القائمة (٩-٦): يوضح الـ Denmark.dem الـ format للـ ask text

```
Phone | 48140001 | 00 09 99 99;1;_
Phone with Country | 48140001 | \+45 00 00 00 00;1;_
Social Security | 1234567890 | 000000\ -0000;1;_
Zip Code | 3450 | 0000;1;_
Zip Code with Country | DK3450 | !>LL\ -0000;1;_
Date | 260195 | 90\90\1\900;1;_
Date (Windows) | 260195 | 90/90/00;1;_
Long Time | 210515 | !90:00:00;1;_
Short Time | 1345 | !90:00;1;_
```

للحصول على جدول بالرموز المكتوبة التي يمكنك استخدامها لإنشاء masks، أبحث عن الـ MaskEdit في الـ online help الخاصة بـ Delphi. يعد الـ EditMask string من الرموز التي تحدد الرموز الحرفية، مجموعات رموز

الادخال، وشفرات التحكم. يتكون ال mask من ثلاثة اجزاء منفصلة بفصلة منقوطة:

- رمزاً واحداً أو أكثر يحدد مجموعاً رموز الادخال المسموحة.
- الرقم 1 لحفظ الرموز الحرفية فى الادخال - hyphens فى رقم هاتف مثلاً- أو الرقم (0) حتى لا يحفظ الرموز الحرفية.
- رمزاً لعرض للاماكن الخالية فى نافذة ال MaskEdit- هذا يكون عادة underscore أو blank.

توضح EditMask string كيفية إنشاء كل جزء من ال mask الكامل. ها هو ال mask:

!\\(999\\)000\\-0000;1;\_

فى وقت التشغيل، يجعل هذا ال mask ال MaskEdit يعرض قالب الادخال التالية:

( ) \_ \_ - \_ \_

فى ال mask string، تخبر نقطة تعجب رأسية ال control ان يطرح جانباً المسافات المتتالية من الادخال الناتج. اذا لم تبدأ ال mask بنقطة تعجب، يتم المحافظة على المسافات المتتالية. اكتب back slash الى الخلف لتحديد رمزاً حرفياً- هناك ثلاثة فى هذا المثال: أقواس الفتح والإغلاق حول code مساحة الأرقام الثلاث الخاصة بالرقم، و hyphen التى تفصل ال exchange والرقم. يظهر ال control هذه الرموز فى نافذة ال edit، وتمر عليها text cursor مروراً خاطفاً اثناء الكتابة تلقائياً.

وال 9 فى ال mask تشير الى ان المستخدمين بإمكانهم إدخال رقماً أولاً شئ فى هذا المكان. استخدام ال 0 لتطلب رقماً. استخدم حرف ال A الكبير للسماح بالمدخل الهجائى الرقمى؛ استخدم حرف ال C الكبير لتطلب رمزاً. انظر online help الخاصة بـ Delphi لتعرف رموز ال mask اخرى.

والجزء الثانى من edit mask (1; فى المثال السابق) يشير اذا ما تحفظ الرموز الحرفية فى الادخال الناتج. أدخل 0; الى الرموز الحرفية المتروكة. مع هذا الخيار،

## الباب التاسع : العمل مع Single-Line Strings

يتلقى البرنامج رقم هاتف مثل الـ 1212-555(800) كـ string للارقام 8005551212.

والجزء الثالث والاخير من edit mask تحدد الرمز الذى يجب عرضه لمواضع الادخال فى قالب الـ mask عادة، يجب ان تحدد underscore أو مسافة خالية، ولكن يمكنك استخدام رمزاً آخر اذا اردت. على سبيل المثال، قم بتعيين الـ underscore المتابعة الخاصة بالـ mask السابق الى نقاط.

تعتبر النقاط ورموز علامات الترقيم الاخرى Fonts متناسبة تنتهى بقوالب الـ MaskEdit التى تشغل مسافة اقل من الادخال النهائى. للحصول على قالب ذا شكل افضل عند استخدام النقطة كرمز موضع الادخال، اختر Font احدى المسافة مثل الـ Courier New للـ MaskEdit control.

يمكنك إنشاء ملفات الـ dem. الخاصة بك لقوالب الـ mask للتحميل فى الـ Input Mask Editor dialog الخاص بـ Delphi. وهذا هو مجرد ملف نص عادى ذو امتداد اسم الملف dem. يتكون كل سطر فى الملف من ثلاثة اجزاء: اسم الـ mask، inputs، و edit-mask string. على سبيل المثال، ها هو مدخل الـ Zip Code من الـ Denmark.dem:

Zip Code | 3450 | 0000;1;\_

**ملحوظة:** يقدم Delphi ملفات dem. لمختلف البلدان واللغات المحلية. تحتوى هذه الملفات على mask للادخال تتبع الـ format الخاصة بالدول. للعثور على الملفات، إبحث فى دليل التركيب \bin الخاص بـ Delphi.

### الـ Password entry :

يمكنك استخدام الـ Edit أو MaskEdit لتوفير password-entry box (وهذا ايضاً ينطبق على الـ DBeedit لقاعدة البيانات). قم بتصميم أى رمز غير الـ #0 فى خاصية الـ PasswordChar. على سبيل المثال، أدخل \* أو # فى هذه الخاصية فى الـ Edit و قم بتشغيل البرنامج. يمكن للمستخدمين إدخال نص فى الـ control، ولكن رمز كلمة المرور فقط هو الذى يظهر على الشاشة.

أعط المستخدمين دائماً الفرصة لتأكيد كلمة المرور بكتابتها مرتين . ان واحدة من اسهل الطرق لفعل هذا هى بإضافة اثنين من ل Edit أو MaskEdit على ال form . وهذه ال format قد يكون dialoge يعرض أمر ، أو يظهره البرنامج عند البدء قبل السماح بالتوصل الى المعلومات .

قم بإنشاء ال OnCloseQuery الخاص بال form . تعتبر القائمة (٩-٧) مثالاً على برنامج يقارن بين ال Edit controls . يسمح البرنامج للform بالإغلاق فقط اذا كانت كلمتى المرور متناسبتين . عند استخدام هذا البرنامج استخدم اثنين من ال Edit وال OnCloseQuery لمنع ال form من الإغلاق الا اذا كانت كلمتى السر فى ال controls متشابهتان .

#### القائمة (٩-٧):مقارنة اثنين من ال Edit controls

. TS: The following line of code exceeds 64 characters in length. Please break the line. Thanks. - DSprocedure TForm1.FormCloseQuery(Sender: TObject;

```
var CanClose: Boolean);
begin
    if Edit1.Text <> Edit2.Text then
    begin
        ShowMessage('Password incorrect!');
        CanClose := False;
        Edit1.SetFocus;
    end;
end;
```

أو، يمكنك إدخال Password Dialog form فى تطبيق . إتبع هذه الخطوات لتجربة هذا القالب :

١- ابدأ مشروعاً جديداً، اختر FileNew Form أو اضغط زر ال New . اختر Password Dialog من ال Browse Gallery dialog .

٢- أضف Button على ال form الرئيسية . اضغط الزر مرتين وادخل هذه العبارات فى ال OnClick الخاص بال Button :

```
PasswordDlg.ShowModal;
ShowMessage>PasswordDlg.Password.Text);
```

## الباب التاسع : العمل مع Single-Line Strings

٣- أضف امر الuses الخاص بال Unit2 الى ذلك الخاص بال Unit1 قرب قمة source code module الخاصة ب Unit1 .

٤- قم بتشغيل البرنامج واضغط الزر . أدخل كلمة المرور في dialog box . عندما تضغط OK يعرض البرنامج مدخلاتك في ShowMessage dialog . لاحظ ان البرنامج المضيف يتوصل الى نص كلمة المرور الخاص بال dialog على انها التعبير PasswordDlg.Password.Text . (تعتبر كلمة المرور Edit object تمتلكه ال PasswordDlg).

### افكار للمستخدم الخبير

● للحفاظ على مسافات التخزين ، ضع قبل string function parameters و ال procedure كلمة const . هذا يخبر Delphi ان ال procedure يستخدم ال string ولكنه لا يغيره . وبالتالي ، يمكن ان يمرر Delphi ال string ، الذى يهدر مسافات التخزين . (ان ال string ليست ذاكرة تخزين مخصصة فى النسخة الحالية من Delphi ، ولكن ال const لا يزال يحفظ string references خفى فى ال try-finally ، والتى يمكن ان تنشئ اطارات تخزين كبيرة) . على سبيل المثال ، عرف procedure كما يلى لإنشاء string parameter ثابت :

```
procedure MySubroutine(const S: String);
begin
end;
```

● حدد ال ShowAccelChar الخاصة بال Label ب False لإبطال قدرة ال Label على تزويد accelerator key لل controls الاخرى . قد تريد ان تفعل هذا ، مثلاً ، لعرض علامة ال & فى ال Caption الخاص بال Label . ولكن يمكنك عرض ال & ببساطة بكتابة إثنين منها . ويتم عرض ال Label Caption && على انه & .

● لقصر الادخال من خلال ال Edit control على حروف الكبيرة أو الحروف الصغيرة ، بدلاً من كتابة ال OnKeyDown event الخاص بال control ،

اختر قيمة لخاصية ال CharCase . يمكنك تحديد ال CharCase بـ ecLowerCase ، أو ecNormal ، أو ecUpperCase .

● ان ال Label ليس لها window handles ولذا فهي تستهلك ذاكرة اقل من ال Edit . للحفاظ على الذاكرة ، استخدم دائماً ال ال Label بدلاً من ال Edit لنص القراءة فقط .

● يعرض ال MessageDlg وال MessageDlgPos عادة glyph bitmaps على ازرارهم . اذا لم تكن تريد glyphs على الازرار ، أدخل العبارة التالية في ال OnCreate الخاص بال form الرئيسية :

MsgDlgGlyphs := False;

● حدد خاصية ال Transparent ال Label بـ True لعرض text labels فوق الصور الجرافيكية مثل ال bitmaps .

● ان ال Label لا يعد windowed control ، ولكونه هكذا ، فلا يمكن ان يظهر فوق windowed control مثل ال Button . اذا لم يكن بإمكانك ان تجبر ال Label على الظهور فوق component آخر باوامر ال EditlBring to Front أو EditlSend to Back ، فإنك تحاول المستحيل . جرب وسيلة اخرى للحصول على التأثير الذي تريده ، أو استخدم ال TStaticText component .

● ان خاصية ال Modified التابعة لـ Edit component ، وهي قيمة Boolean محددة بـ True أو False ، تشير اذا ما حدث اية تغييرات لمحتويات ال control منذ إنشاء ال Edit أو منذ تحديد ال Modified بـ False . بعد قبول ال ادخال من ال Edit ، حدد ال Modified بـ False . يمكنك حينئذ فحص هذه الخاصية لتحديد ما اذا كانت محتويات ال Edit قد تغيرت .

● يمكنك التوصل الى نص ال MaskEdit باستخدام خاصيتين ، ال EditText أو ال Text . اذا كان object's mask يحدد ان الرموز الحرفية يجب ان يتم حفظها ، فإن ال EditText و ال Text لهما المحتويات المتطابقة ، ويمكنك استخدام أى منهما للحصول على control's text . اذا لم يكن ال mask يحدد حفظ الرموز الحرفية ، فإن ال EditText تمثل النص الذي يراه المستخدمون في نافذة ال MaskEdit ، و ال Text تمثل نفس هذا النص ناقص رموزه الحرفية .

## الباب التاسع : العمل مع Single-Line Strings

● لقصر الادخال من خلال الـ MaskEdit على الحروف الكبيرة والحروف الصغيرة، حدد خاصية الـ CharCase بـ ecLowerCase أو ecUpperCase .  
للسماح بالادخال بالاحرف الصغيرة والحروف الكبيرة، حدد الـ CharCase بقيمتها الافتراضية ، ecNormal . قد يكون هذا ابسط من تحديد دخول الحروف الصغيرة وحروف الكبيرة باستخدام رموز الـ mask .

● عند تعاقب الـ strings ، تذكر ان كل استخدام لعامل الزائد يستهلك ذاكرة للتخزين المؤقت . فى كثير من الحالات ، يمكنك استخدام الـ Format function لتقليل استخدام الذاكرة . على سبيل المثال ، لتعاقب ثلاث strings A ، B ، C ، وتعين الناتج الى S ، استخدم عبارة مثل :

`S := Format('%s%s%s', [A, B, C]);`

وهذا هو ما يعادله منطقياً للعبارة التالية :

`S := A + B + C;`



## المشروعات التي يمكنك تجربتها

(٩-١) : قم بتصميم module تعرض رسائل خطأ باستخدام الـ MessageDlg أو الـ MessageDlgPos .

(٩-٢) : اكتب برنامجاً اختيارياً يسمح للمستخدمين ببرمجة accelerator hot keys لواحد أو أكثر من الـ Edit objects .

(٩-٣) : قم بإنشاء MaskEdit objects لإدخال عناوين البريد الالكتروني لخدمات عديدة مثل الـ Internet والـ CompuServe . احفظ قوالبك فى ملف نص dem .

(٩-٤) : اكتب password entry dialog أو قم بالتحسين فى قالب الـ Password Dialog الخاص بـ Delphi . يجب ان يقوم قالبك بتوفير السبل للمستخدمين ليؤكدوا كلمة المرور وذلك بإدخالها مرتين .



## ملخص:

ان ال Label، وال Edit، وال MaskEdit components الخاصة بـ Delphi يمكن ان تتعامل مع غالبية متطلبات single-line string بالتطبيق.

- يقدم Delphi نوعين اساسيين من الـ strings-AnsiString وال ShortString. ان نوع بيانات الـ strings يرجع اصلاً الى الـ AnsiString الا اذا كان يستخدم خياراً {H-}، في هذه الحالة تكون الـ Strings من نوع الـ ShortString. تعتبر الـ Pascal strings ذات الـ ٢٥٥ رمزاً بطول البايت مساوية للـ ShortString. ان انواع بيانات الـ String (و الـ AnsiString) تنشئ strings ديناميكية غير محددة الطول. يتم التحكم الآلى في الذاكرة عند استخدام هذه الانواع من الـ strings. ان {H-}، يعتبر مفيداً فقط لإجراء عملية الـ compiling للـ Pascal code والـ Delphi القديمة.

- يقدم Delphi ايضاً نوع الـ WideString، الذى يخزن قيم رموز ذات ١٦ بت Unicode. ان strings العريضة يتم استخدامها فى تطبيقات اللغات مثل اليابانية، التى لا يمكن ان تمثل رموزها كاملة باستخدام البايت ذات الـ ٨ بت. وهى ايضاً تسمح بالعرض الغير غامض للنص بلغات متعددة.

- حاول استخدام الـ Pascal Strings لجميع احتياجاتك للتعامل مع الـ strings. الـ functions والـ procedures للـ Pasc string فعالة جداً، ولان الـ string من النوع المركب داخلياً، يمكنك استخدام string فى تعبيرات. على سبيل المثال، لتعاقب إثنين أو أكثر من الـ strings، اضفهم الى بعضهم البعض بعوامل علامة الزائد، كما فى التعبير S1 + S2 + S3. يمكنك ايضاً مقارنة الـ Pascal Strings باستخدام عوامل الـ <، <=، >، >=، و <>. يجب ان تستدعى functions ومثل الـ StrComp لمقارنة الـ null-terminated strings.

- لعرض رسائلك استدع الـ ShowMessage أو الـ ShowMessagePos. لعرض رسائل وحث المستخدمين على ضغط زر- مثلاً، لطلب الأذن بحذف ملف- استدع الـ MessageDlg أو الـ MessageDlgPos. يمكنك ايضاً استدعاء الـ MessageBox function المعيارية بالـ Windows، ولكن هذا يتطلب منك استخدام الـ null-terminated strings.

## الباب التاسع : العمل مع Single-Line Strings

● لحث المستخدمين على ادخال بيانات نص احادى السطر ، استدع ال InputBox أو ال InputQuery. هذه ال functions يمكن ان تتعامل مع اغلب إحتياجات الادخال بسطر واحد ، وتوفر عليك الوقت والجهد فى تطوير input dialogs الخاصة بك مع ال Edit component.

● كما تعرف غالباً يمكنك استخدام ال Label object وعناصر النص الثابت الاخرى فى ال forms. ولكن ال Label object يمكن ايضاً ان يوفر accelerator hot key control آخر مثل ال Edit أو ال MaskEdit الذين لا يقدمون اختيار مفتاح Alt.

● لاحتياجات ادخال السطر الواحد العامة ، استخدم ال Edit object. وهذا ال components له events وخصائص عديدة يمكنك استخدامها لتوفير مناطق ادخال. لل Edit عبارة عن text editor محرر نص احادى السطر ، ويدعم تلقائياً عمليات الفأرة ولوحة المفاتيح ، بما فى ذلك القص ، النسخ ، واللصق.

● حاول ان تجعل تطبيقاتك قابلة للعمل بالفأرة ولوحة المفاتيح. على سبيل المثال ، قم بتشغيل تطبيق ال TelName ، والذي يظهر تقنيات جهاز لوحة المفاتيح مثل تحديد ال focus عند تلقى ضغطه زر.

● استخدم ال MaskEdit لتوفير ال template-based Edit controls. ادخل أو اختر mask لخاصية ال EditMask ل MaskEdit object لتطلب من المستخدمين ، مثلاً ادخال رقم هاتف أو تاريخ باستخدام form معينة.

● لإدخال كلمة المرور ، عين رمزاً لخاصية ال PasswordChar ل Edit أو ال MaskEdit object (وهذا ايضاً يعمل مع ال DB-Edit components). تعرض ال objects هذا الرمز فقط لكل المدخلات.

لقد بدأت الآن فقط تتعرف على إمكانيات الادخال والإخراج الخاصة بـ Delphi. فى الباب التالى ، سوف تفحص components للتحكم فى ال text objects متعدد السطور.

\*\*\*\*\*



# الباب العاشر

## العمل مع نص متعدد السطور

محتويات هذا الباب:

• Components

• ال Memos

• Text and the clipboard

• Scrolling down the river

• StringGrids

سوف يبدأ المبرمجون، آجلاً أو عاجلاً، في كتابة text editors الشخصي الخاص بهم بكل الميزات التي يحلمون باستخدامها. ان تصميم text editors بسيط بأقل قدر من اوامر نص، قص ولصق السطور، وقراءة وكتابة الملفات كل هذا يعد أمراً بالغ المشقة. وانا لم اكمل ابدأ text editor الخاص بي، واتمنى ألا اشرع في إنهاءه.

ولحسن الحظ، مع ال multiple-Line text components الخاص بـ Delphi، إنني لا احتاج غالباً الى العمل على هذا ال code مرة اخرى. ان Memo واحد يوفر كل إمكانيات ال Windows Notepad editor. فقط ادخل ال Memo في form وحرك عصاك السحرية لتحصل على text editor على الفور.

في هذا الباب، إنك تستخدم ال Memo وال StringGrid لإقامة text objects متعدد السطور في واجهة تطبيق المستخدم لتطبيقك. وسوف اشرح ايضاً موضوعات متعلقة بهذا ايضاً مثل إنتقال المعلومات خلال ال clipboard وكيفية استخدام ال ScrollBar وال ScrollBox، والالذان سوف تحتاجها دائماً عند العمل بنص متعدد السطور.

## : Components

فيما يلي Delphi components للعمل بنص متعدد السطور:

• **Memo**: من الناحية العملية برنامج معالجة كلمات، يضيف ال Memo إمكانات تشبه ال Notepad الى التطبيق. ان متصفح ملف نص ال Readme الخاص بهذا الباب- والذي يمكنك استدعاه من تطبيق آخر installation utility- يوضح كيفية استخدام ال Memo objects. ال Standard: Palette.

• **ScrollBar**: هذا ال component يضيف scroll bars لل form، ويمكنه ان يعمل مستقلاً ك range-selection control. على سبيل المثال، فى هذا الباب، انك تستخدم ال ScrollBar لإنشاء color selection dialog. ال Standard: Palette.

• **ScrollBar**: يمكنك ان تسمى هذا ال component بال object الحاوى ذى العجلات. استخدم ال ScrollBox لإنشاء scrollable objects، text displays، panels، options dialogs، وعناصر بيئة معقدة اخرى تحتاج الى إمكانات scrolling. ال Additional: Palette.

• **StringGrid**: هذا ال component ذو القدرة العالية، والذي تم تقديمه فى الباب السابق، ينظم string lists فى ال form فى صورة اعمدة وصفوف يمكن التوصل إليها ك array ثنائى الابعاد. مع هذا ال component، لديك اغلب العناصر البيئية اللازمة لإنشاء جداول ودفاتر حسابات. ان تطبيق ال CharGrid الخاص بهذا الباب، والذي يشبه ال Windows Character Map utility، يوضح كيفية استخدام ال StringGrid objects لإنشاء دفاتر حسابات قابلة للاختيار. ال Additional: Palette.

### بفضل ال Memos:

يغلف ال Memos ال Edit control متعدد السطور المعيارى بال Windows. ولكن ال Memos ليس control قديماً فى زى جديد- إنه text editor معقد متعدد السطور يمثل ميزات object-oriented programming. على سبيل المثال، ان ال Edit control المعيارى بال Windows يجعل التحكم فى

## الباب العاشر : العمل مع نص متعدد السطور

حواجز تخزين النص امر صعب وميال للخطأ . ان الـ Memos ، الذى يغلف الـ Edit control فى الـ Delphi class (TMemo)، ويهزم هذه الحواجز الـ methods المصممة بذكاء للتوصل الى نص متعدد الاسطر فى forms حواجز تخزين أو string-list .

### إنشاء Memos للقراءة فقط:

ان الـ Read-only Memo مفيدة فى عرض المعلومات . على سبيل المثال ، انظر تطبيق الـ Fancy فى الباب السادس ، الذى يستخدم إثنين من الـ Memos لعرض قيم خاصية لأنماط الـ Panel 3D المتنوعة . ان استخدام Memos لقوائم القراءة فقط اسهل بكثير من إنشاء Labels متعددة .

لإنشاء Memos للقراءة فقط ، حدد ثلاث خصائص للقيم الواردة فى جدول (١٠-١) .

جدول (١٠-١): خصائص الـ Memos للقراءة فقط

المواصفات	القيمة	الوصف
Enabled	False	تمنع المستخدمون من اختيار نص فى نافذة الـ Memo
TabSrep	False	وتمنع المستخدمون من ضغط الـ Memo
ReadOnly	True	تمنع المستخدمون من عمل أية تغييرات أو إدخال نصاً جديداً فى نص الـ Memo

بعد تحديد هذه القيم ، عين نصاً لخاصية الـ Lines لـ Memo . اضغط الزر البيضاوى لفتح محور قائمة الـ String الخاصة بـ Delphi ، واكتب أو انسخ والصق نصك .

فى النسخ ذات الـ 16-bit من Delphi ، يمكن أن يحمل الـ Memo الى 32K من النص . كل سطر فى الـ Memo لا يمكن أن يكون اطول من 1,024 رمزاً ، شريطة ألا يكون عرض السطر المعروض اطول من 30,000 pixels ومواصفات الـ Windows هذه تم إزالتها من النسخ ذات الـ 32-bit من Delphi ، ويمكن الآن أن تحمل الـ Memo كم غير محدد من النص ، بالرغم من أن الـ Windows 95 مازال يحدد الكم الإجمالى بـ 64K .

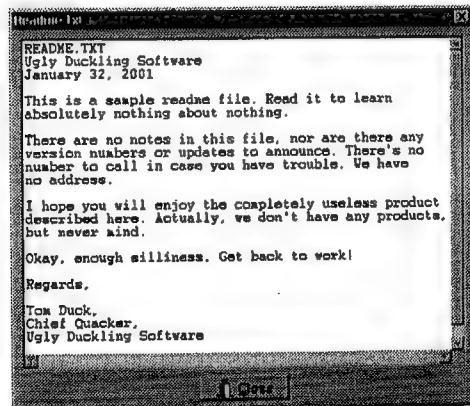
## إخراج وإدخال ملف النص:

تعتبر خاصية الـ Lines للـ TStrings object Memo، ولذلك، يمكنك استخدام تقنيات string-list من الباب الثامن لقراءة، وكتابة، وتعديل نص الـ Memo. على سبيل المثال، هذا يعرض السطر الأول في الـ Memo:

```
ShowMessage(Memo1.Lines[0]);
```

استدع الـ LoadFromFile method الخاص بالـ Lines لقراءة ملف النص في الـ Memo. استدع الـ SaveToFile لكتابة محتويات الـ Memo على قرص. الـ procedures، يمكنهما إقامة الـ editor الشبيه بالـ Notepad الخاص بك.

**على القرص المدمج:** كمثال على التعامل مع ملف نص، يعرض تطبيق الـ Readme نسخة قراءة فقط لملف نص في الـ Memo يمكن لبرنامج آخر - installation utility، مثلاً - أن يستدعيه لعرض ملاحظات حول التطبيق بدلاً من استخدام الـ method القديم وهو استدعاء الـ Notepad، والذي يمكن المستخدمين من تغيير ملف القراءة فقط الخاص ببرنامجك. بعد قائمة الـ Readme، سوف اشرح كيفية تشغيل البرنامج من داخل تطبيق آخر لـ Delphi - مثلاً، يمكنك استخدام التقنية في برنامج installer. يوضح شكل (١٠-١) عرض الـ Readme. توضح القائمة (١٠-١) الـ source code، الموجودة على القرص المدمج في دليل الـ Source\Readme.



شكل (١٠-١): يوضح تطبيق الـ Readme كيفية إنشاء متصفح ملف نص قراءة فقط

القائمة (١٠-١) : Readme\Main.pas

```
unit Main;

interface

uses
  Windows, SysUtils, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, Buttons;

type
  TMainForm = class(TForm)
    Memo1: TMemo;
    BitBtn1: TBitBtn;
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FormActivate(Sender: TObject);
var
  FileName: string;
begin
  if ParamCount >= 1
  then FileName := ParamStr(1)
```



```

else FileName := 'Readme.Txt';
Memo1.Lines.LoadFromFile(FileName);
Caption := FileName;
end;

end.

```

يوضح OnActivate form الرئيسية لل Readme كيفية الحصول على parameter تم تمريره من عملية أخرى الى تطبيق . (أدخل parameters إختبارية بأمر ال RunParameters). إن المتغير العام ParamCount يساوى عدد parameters ، كلاً منها متاح من خلال ال ParamStr function . قم بتمرير ال parameter indexed الى ال ParamStr . وال parameter indexed بـ (0) يساوى اسم مسار التطبيق - قم بعرضه فى ال OnClick الخاص بال Button بعبارة :

```
ShowMessage(ParamStr(0)); { Show application pathname }
```

فى ال Readme ، اذا كان ال ParamCount أكبر من أو يساوى واحد ، يعين البرنامج parameter قد تم تمريره الى ال FileName string ، وإلا ، فإنه يحدد ال FileName للملف الافتراضى ، بـ Readme.Txt وتقوم عبارة واحدة بتحميل نص هذا الملف فى ال Memo :

```
Memo1.Lines.LoadFromFile(FileName);
```

بعد ذلك ، يحدد البرنامج ال Caption الخاص بال form بـ FileName string . عند استخدام ال Memo لعرض ملفات نص ، استخدم ال Label ، Caption ، أو text object آخر لعرض اسم المسار حتى يعرف المستخدمون أى الملفات يتصفحون .

بالرغم من أن ال Readme لا يكتب على قرص ، يمكنك أن تفعل هذا بسهولة باستدعاء ال SaveToFile procedure لخاصية ال Lines ل Memo . يقوم ال procedure بإنشاء ملف جديد ، أو ملف يكتب على ملف اذا كان يوجد واحداً يحمل هذا الاسم . للحماية من الكتابة على ملف موجود ، استدع ال FileExists قبل استدعاء ال SaveToFile ، كما فى هذا ال code (بافتراض ان ال FileName يعتبر متغير String) :

```
FileName := 'C:\Anyfile.Txt';
if FileExists(FileName) then
  ShowMessage('File exists')
else
  Memo1.Lines.SaveToFile(FileName);
```

لتمرير ال parameters الى برامج ، استخدم أمر ال Run لزر ال Start بال Windows 95 أو 98 إدخال C:\Path\Readme Filename.txt لتشغيل ال Readme وتمرير ال parameter Filename.txt ، إليه عند تشغيل برامج داخل Delphi ، اختر أمر ال Run\Parameters ، وإدخل واحداً أو أكثر من ال parameters لتمررها الى البرنامج فى المرة التالية التى تشغله فى نمط إزالة الأخطاء .

**على القرص المدمج:** بالرغم من انه بإمكانك تشغيل ال Readme تطبيق مستقل ، لقد صممته على ان يتم استدعائه من برنامج آخر ، مثل installation utility . لعرض هذه التقنية ، التى يمكنك استخدامها لتشغيل أى برنامج تنفذى - فهى تعمل أيضاً للملفات ال DOS PIF - اختر تطبيق ال Readme الموجود على القرص المدمج المرفق بهذا الكتاب فى دليل ال Readme قم بتحميل البرنامج فى Delphi واضغط F9 لتشغيل . إختتر زر ال Click Me لتشغيل تطبيق ال Readme . يوضح شكل (١٠-٢) عرض ال Readme . لتشغيل برنامج ال Readme ، اختر زر ال Click Me [انظر شكل (١٠-١)] . توضح القائمة (١٠-٢) ال source code ، الموجودة فى دليل ال Source\Readme .



**ملحوظة:** اذا لم يعرض ال Readme ملف ال Readme.txt المتوقع ، إتبع هذه الخطوات . قم بتحميل ملف ال Runme.dpr فى Delphi واضغط Ctrl+F9 لإجراء عملية ال compile . ثم قم بتحميل Runme.dpr واضغط F9 . كلا المشروعين موجودان على القرص المدمج فى دليل ال Source\Readme . يجب ان يكون الآن قادراً على ضغط ال Click Me لتشغيل برنامج عرض ال Readme . اذا واجهت مشكلة ، فالسبب غالباً هو ان البرنامج يتوقع ان يجد ال Readme.txt فى الدليل الحالى ، والذي قد تكون قمت



بتغيره اثناء استخدام Delphi. ان إعادة تشغيل كل برنامج عرض يضمن ان ال  
Readme.exe يمكن ان يجد ملف ال Readme.txt.



شكل (١٠-٢): يوضح تطبيق ال Runme كيف  
يمكن لتطبيق ان يقوم بتشغيل برنامج آخر

القائمة (١٠-٢): Readme\Test.pas

unit Test;

interface

uses

Windows, SysUtils, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Buttons;

type

```
TTestForm = class(TForm)
    ClickMeButton: TButton;
    Label1: TLabel;
    Bevel1: TBevel;
    BitBtn1: TBitBtn;
    procedure ClickMeButtonClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;
```

var

```
TestForm: TTestForm;
```

```
implementation
```

```
uses FMXUtils;
```

```
{ $R *.DFM }
```

```
(* 16-bit version; still works but WinExec is obsolete
procedure TTestForm.ClickMeButtonClick(Sender: TObject);
var
    K: Integer; { Result of calling WinExec }
begin
    K := WinExec('Readme.Exe Readme.Txt',
        SW_SHOWNORMAL);
    if K < 32 then
        ShowMessage('Error running README.EXE');
end;
*)
```

```
{ 32-bit version; calls ExecuteFile in FMXUtils unit, which
  is located in Delphi's Demos\Doc\Filmanex folder }
procedure TTestForm.ClickMeButtonClick(Sender: TObject);
var
    H: THandle;
begin
    H := ExecuteFile('Readme.exe', 'Readme.txt', '',
        SW_SHOWNORMAL);
    if Integer(H) < 32 then
        ShowMessage('Error running Readme.exe');
end;
```

```
end.
```

يوضح OnClick الخاص بزر ال Click Me كيفية تشغيل برنامج من داخل آخر. فى النسخ السابقة من Delphi، كانت الطريقة الصحيحة لفعل هذا هى باستدعاء ال WinExec function. هذا مازال يعمل، ولكنه نادر. فى ظل ال Windows 95، و ال Windows 98، و ال Windows NT ذو ال ٣٢ بت، من المفضل ان تستدعى التطبيق ال Windows API ShellExecute function.

ويتطلب فعل هذا مزيداً من العمل أكثر مما تحتاج غالباً. لحسن الحظ، هناك حل ايسر، كما هو موضح فى القائمة. تحتوى ادلة Doc و Demos التابعة لـ Delphi تحتوى على unit منفصلة، وهى FMXUtils.pas، فى مجلد ال Filemanex. اصف هذه unit الى المشروع باستخدام ال ProjectAdd to project. بالإضافة الى ذلك، اصف التعريف التالى الى ال form unit، تحت كلمة ال implementation الخاصة بها مباشرة:

uses FMXUtils;

يمكن ان تستدعى العبارات الآن ال ExecuteFile الخاصة بال unit كما موضح فى التطبيق لتشغيل برنامج آخر. يمكنك حتى تنفيذ ملفات البيانات، التى تفتحها داخل التطبيقات المتعلقة بها. على سبيل المثال، قم بتمرير ملف نص الى ال ExecuteFile لفتح هذا الملف باستخدام التطبيق الافتراضى لنوع الملف- وهو غالباً ال Windows Notepad utility.

**تحذير:** اذا كانت لديك مشكلة فى تشغيل تطبيق ال Runme على القرص المدمج المرفق بهذا الكتاب، فإن السبب غالباً أن FMXUtils.pas الخاص بـ Delphi غير موجود فى المكان الذى يتوقعه فيه البرنامج. استخدم ال Windows Explorer لوضع الملف، ثم استخدم أمر ال ProjectAdd to project لإضافة ال unit لل Runme استخدم أمر ال ProjectRemove from project لإزالة المرجع القديم الى ال FMXUtils.pas.

يتم تعريف ال ExecuteFile function وال parameters الخاصة بها كما يلى:

```
function ExecuteFile(const FileName, Params,
    DefaultDir: string; ShowCmd: Integer): THandle;
```

● **FileName**: يمرر اسم ملف string أو اسم مسسار تام فى هذا الparameter.

● **Params**: يمرر فى هذا الstring أية parameters، خيارات، أو أسماء ملف تريد تمريرها الى ملف الcode فى الFileName. يقوم برنامج العرض بتمرير الReadme.txt فى هذا الparameter.

● **DefaultDir**: حدد هذا الstring بالدليل الذى يبحث فيه الExecuteFile عن الFileName. لاحظ اذا كان الParams هو اسم ملف آخر، فإن هذا التحديد لن يجعل البرنامج المنفذ يبحث فى الدليل المحدد.

● **ShowCmd**: هذا يمكن ان يكون أى ثابت من ثوابت الSW\_SHOW.... مثل الSW\_SHOWNORMAL لعرض نافذة تطبيق فى حالة البدء الطبيعية لها، أو قيمة أخرى مثل الSW\_SHOWMAXIMIZED لفتح النافذة على الشاشة الكاملة عند بدء البرنامج. انظر ملف الWin32.hlp لثوابت الSW\_SHOW الأخرى التى يمكنك تمريرها للExecuteFile.

مثل الWinExec، يقوم الExecuteFile (و الWindows API ShellExecute function)، اذا كان ناجحاً، بادخال قيمة عدد صحيح أكبر من أو تساوى ٣٢ وأى قيمة اقل من ال٣٢ تشير الى وجود خطأ والقيمة المدخلة هى بالفعل الhandle الحالى للتطبيق المستهدف، بالرغم من ان هذه الحقيقة ليست بالغة الأهمية. ان الWinExec البائدة ادخلت هذه القيمة كعدد صحيح، مما جعلها اسهل فى الاستخدام بهذا الحال. و الExecuteFile الجديدة (و الShellExecute) تدخل القيمة الناتجة ك handle. ان فحص قيمة العدد الصحيح الخاص بها يتطلب استخدام تعبير الtype-cast مثل :

```
.if Integer(H) < 32 then
```

```
    ShowMessage('Error running Readme.exe');
```

واخيراً، قم بتجربة التالى . اضعف الـ FMXUtils.pas unit للمشروع، واضف FMXUtils uses تحت كلمة implementation مباشرة، ثم أضف Button object على form قم ببرمجة الـ Button كما يلى :

```

procedure TForm1.Button1Click(Sender: TObject);
var
    H: THandle;
begin
    H := ExecuteFile('C:\Windows\Sol', "", SW_SHOWNORMAL);
end;
```

يتم تمرير strings (empty) null للـ string parameters الغير مستخدمة . اضغط F9 لتشغيل البرنامج، ثم اضغط الزر لتنفيذ لعبة الـ Solitaire الخاصة بالـ Windows .

فيما يلى الـ code العائدة ذات الـ 16-bit التى تقوم بنفس الشئ فى الـ Windows 3.1 . وهذه مازالت تعمل، ولكن يجب استبدالها بالبرمجة الجديدة الموجودة فى المثال السابق :

```

{ This is the obsolete 16-bit method.
  Don't use this technique any longer. }
procedure TForm1.Button1Click(Sender: TObject);
var
    K: Integer;
begin
    K := WinExec('C:\Windows\Sol', SW_SHOWMAXIMIZED);
end;
```

### إدارة النص فى الـ Memo objects الخاصة بك:

من الطبيعى أن تحتاج الى إضافة، حذف، إدخال وتعديل النص فى الـ Memo الخاصة بك . إن برمجة هذه المهام ومهام أخرى يتطلب فهم جيد لكيف توفر الـ Memo الوصول الى سطور النص الخاص بهم . يمكنك الوصول الى نص الـ Memo :

\* كأنه قيمة الـ string لخاصية الـ Text، والتى لا تظهر فى الـ Object Inspector وتعتبر متوفرة فقط فى وقت التشغيل . استخدم هذه الخاصية فقط

## الباب العاشر : العمل مع نص متعدد السطور

عندما تريد استغلال نص الـ Memo كأنه objects . لم تعد قيمة هذه الخاصية محددة بـ ٢٥٥ رمزاً كما كانت في الأصل في النسخة الأولى من Delphi .

\* كقاعدة من strings في خاصية الـ Lines ، باستخدام الـ methods الموضحة في الباب التاسع لـ TStrings objects والـ TStringList . بشكل عام ، تعبير هذه هي أكثر الطرق تنوعاً للتوصل الى نص الـ Memo وهي دائماً الأفضل عندما تحتاج أن تعالج سطور النص بمفردك .

لعرض واستخدام نص الـ Memo على أنه string فردي ، استخدم خاصية الـ Text . وهي تعتبر موروثه من الـ TControl ولكنها غير مذكورة في نافذة الـ Object Inspector . على سبيل المثال ، أضف Button و Memo على from ، واستخدم هذه العبارة في الـ OnClick الخاص بالـ Button لعرض نص الـ Memo :

```
ShowMessage(Memo1.Text);
```

إلصق بعض النص في نافذة الـ Memo1 واضغط الزر . وكأن هذا يستخدم لإزالة محتويات الـ Memo ليكون الحد الأقصى لها ٢٥٥ رمزاً ، ولكن الآن تستخدم الـ Delphis components تلك ، نوع الـ String datas الديناميكي ، ولم تعد هذه المشكلة تحدث .

للتوصل الى سطور فردية في الـ Memo ، استخدم خاصية الـ Lines . ولأن هذا يعد object من نوع الـ TStrings ، يمكنك نقل النص بين objects أخرى لها خصائص الـ TStrings . على سبيل المثال ، استخدم عبارة مثل التالية لنسخ الـ Items الخاصة بالـ ListBox الى الـ Lines التابعة للـ Memo :

```
Memo1.Lines := ListBox1.Items;
```

قد تستخدم التقنية العكسية لتمكين المستخدمين من إدخال بنود الـ ListBox باستخدام خصائص الـ text-editing في الـ Memo . قم بأداء التعيين في الاتجاه الآخر ، من الممكن أن يكون في الـ OnClick الخاص بالـ Button :

```
ListBox1.Items := Memo1.Lines;
```

لإدخال نص جديد في الـ Memo ، يمكنك تعيين strings لخاصية الـ Text . ولكن كن على حذر من هذه الطريقة - فهي تحل محل محتويات الـ Memo بأكملها ، بغض النظر عن الحجم :



```
Memo1.Text := 'I'd rather be sailing!';
```

إن علامتى التنصيص الفرديتين تنشئ علامة الفصلة العلوية الدالة على حذف حرف (") داخل الـ string. بالرغم من أن العبارة السابقة تعمل بشكل جيد، ولكن من الأفضل أن تستخدم خاصية الـ Lines. هذه العبارة تضيف string لقائمة الـ Lines:

```
Memo1.Lines.Add('Bananas are not Oranges');
```

امسح محتويات الـ Memo باستدعاء الـ Clear method كهذا:

```
Memo1.Lines.Clear;
```

ولكن لأن الـ Memo له الـ Clear method الخاص به، يمكنك استدعاء الـ Clear دون الإشارة إلى الـ Lines. هذه العبارة مساوية للعبارة السابقة:

```
Memo1.Clear;
```

حدد إذا ما كان الـ Memo خالياً بفحص خاصية الـ Count للـ Lines:

```
if Memo1.Lines.Count = 0 then
```

```
  ShowMessage('You just clicked this button!')
```

```
else
```

```
  Memo1.Clear;
```

### الوصول إلى الـ Memo text buffer:

كما ذكرت، يمكنك، في أغلب الحالات، التوصل إلى نص الـ Memo بسهولة من خلال الـ Lines object الخاص به. ولكن، إذا كنت تحتاج أن تؤدي عمليات مباشرة على text buffers، يمكنك التوصل لنص الـ Memo على أنه array متجه من الـ Char. قد يكون هذا مفيداً في بعض التطبيقات، بالرغم من أنه مع نوع بيانات الـ String الغير محددة الطول للـ Object Pascal، لم تعد التقنيات ذات قيمة عالية كما كانت في الماضي. رغم ذلك، فإن هذا الفصل يوضح الأسس. حتى إذا لم تكن تستخدم هذه الطرق، يمكن أن يساعدك هذا الفصل في تحديث التطبيقات القديمة التي تستخدم null-terminated strings مع الـ Memo objects.

## الباب العاشر : العمل مع نص متعدد السطور

هناك خمسة methods للعمل مع نص ال Memo على أنه ال Char array - أى أنه null-terminated strings . وال methods بالترتيب الذى أوردته هو :

● **SetTextBuf**: ينسخ النص فى ال Memo وهذا مسار لرسالة ال `WM_SetText`.

● **SetSelTextBuf**: يحل النص المختار فى ال Memo بنص جديد وهذا مساو لرسالة ال `EM_REPLACESEL` message .

● **GetTextLen**: يحدد عدد الرموز فى ال Memo buffer وهذه القيمة لا تشمل الإلغاء الذى ينهى ال buffer وهذا مساو لرسالة ال `WM_GetTextLength`.

● **GetTextBuf**: ينسخ نص ال Memo الى ال Char array لبرنامج أو ال PChar buffer وهو مساو لرسالة ال `WM_GetText`.

● **GetSelTextBuf**: ينسخ النص المختار من ال Memo الى برنامج Char buffer array أو ال PChar buffer مساو لرسالة ال `EM_GETSEL` يتبعها نسخة string للنص المختار بال Windows .

يمكنك استخدام هذه ال methods مع arrays من ال Char Bytes ، أو مع ال PChar pointers . يمكنك تمرير ال Char array الى ال PChar . على سبيل المثال ، مع هذا التعريف التالى ، يمكنك تمرير إما S أو P الى أى PChar parameter :

var

S: array[0 .. 128] of Char;

P: PChar;

هذه العبارات تنسخ ال String المدعو P الى ال Memo text buffer (لقد افترضت أنك قد بدأت ال strings فى مكان آخر).

Memo1.SetTextBuf(S); { Copy array S to Memo1 }

Memo1.SetTextBuf(P); { Copy string at P to Memo1 }

فى الشرح التالى اننى استخدم الـ PChar pointers لانها أكثر فعالية، خاصة للـ text buffer الكبير. لكن، نفس هذه التقنيات تعمل مع الـ Char array. توضيح القائمة (١٠-٣) كيفية استخدام الـ SetTextBuf لنسخ الـ null-terminated string فى الـ Memo. (فى حالة اذا اردت ان تجرب الـ code، فهى فى الـ OnClick لـ Button). أولاً، يخصص الـ StrAlloc ذاكرة لـ string ذى ١٢٨ رمزاً. ثم، ينسخ الـ StrCopy النص الحرفى الذاكرة. وتقوم التعبيرات #13#10 بإدخال الـ control codes على سطور جديدة فى نهايات للاسطر الثلاثة الأولى. ينسخ الـ SetTextBuf الـ string فى الـ Memo1. بعد ذلك، يحذف الـ StrDispose الذاكرة المخصصة.

القائمة (١٠-٢): يوضح الـ OnClick هذا كيفية إدخال فى الـ Memo

```

procedure TForm1.Button1Click(Sender: TObject);
var
  P: PChar;
begin
  P := StrAlloc(128);
  try
    StrCopy(P,
      'Red skies at night,#13#10 +
      ' Sailor"s delight,#13#10 +
      'Red skies at morning,#13#10 +
      ' Sailor take warning. ');
    Memo1.SetTextBuf(P);
  finally
    StrDispose(P);
  end;
end;

```

بالرغم من اننى اريد ان اوضح هنا كيفية استخدام الـ PChar و text buffers، من المهم ان تفهم انه مع نوع بيانات الـ String لـ Object Pascal، لم تعد البرمجة كتلك الموجودة فى القائمة (١٠-٣) ضرورية.

## الباب العاشر : العمل مع نص متعدد السطور

يمكن استبدال الـ procedure السابق بما يلي ، والذي يعد اسهل فى البرمجة ويعمل بشكل رائع . وبالإضافة الى ذلك ، فهو ايضاً أكثر أماناً لأنه لا يوجد احتمال نسيان التخلص من الـ text buffer الذى تم إنشاؤه باستدعاء الـ StrAlloc :

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
```

```
  S: String;
```

```
begin
```

```
  S :=
```

```
    'Red skies at night,#13#10 +
```

```
    ' Sailor's delight;#13#10 +
```

```
    'Red skies at morning,#13#10 +
```

```
    ' Sailor take warning.';
```

```
  Memo1.Text := S;
```

```
end;
```

تتحكم الـ Memo (وكذلك الـ Edit والـ text components الأخرى) فى ذاكرة تخزين الـ string الخاصة بهم . على سبيل المثال ، عندما تستدعى الـ SetText ، ينسخ Memo بيانات الـ string فى الذاكرة الخاصة به . وإنها لمسئولتك ان تحذف الـ string الديناميكية التى تقوم بإنشاءها ، حتى بعد ان تعينها للـ Memo .

استدع الـ SetSelTextBuf ليحل محل النص المختار (البارز) فى الـ Memo . على سبيل المثال ، قم بتشغيل برنامج اختياري مع الـ event handler الموجود فى القائمة السابقة ، ثم أدخل الـ procedureOnClick آخر للـ Button بالـ code الموجودة فى القائمة (١٠-٤) لتحل محل النص المختار برموز جديدة فى الـ Memo باستدعاء الـ SetSelTextBuf اختر بعض أو كل النص فى الـ Memo ، واضغط الزر الثانى لتستبدل النص بالنص الجديد .

القائمة (١٠-٤) : OnClick لاستبدال النص المختار برموز جديدة فى الـ Memo

```
procedure TForm1.Button2Click(Sender: TObject);
```

```
var
```

```
  P: PChar;
```

```
begin
  P := StrAlloc(128);
  try
    StrCopy(P,
      'How happy is the sailor's life,'#13#10 +
      'From coast to coast to roam;'#13#10 +
      'In every port he finds a wife,'#13#10 +
      'In every land a home.');
```

Memo1.SetSelTextBuf(P);

```
finally
  StrDispose(P);
end;
end;
```

إذا لم يكن هناك نصاً مختاراً في الـ Memo، يقوم الـ SetSelTextBuf بإدخال النص الجديد في الموضع الحالي للـ cursor . لاستبدال كل النص تماماً في الـ Memo، استدع الـ SelectAll methods قبل استدعاء الـ SetSelTextBuf .

لنسخ النص من الـ Memo الى الـ Char array، استدع الـ GetTextBuf أو الـ GetSelTextBuf حدد الحجم المطلوب للـ buffer باستدعاء الـ GetTextLen، قم بتخصيص كل هذه الذاكرة زائد بايت واحد لـ null terminator، واستدع الـ GetTextBuf لنسخ نص الـ Memo في المساحة المخصصة. استدع الـ GetSelTextBuf لنسخ النص المختار فقط الى الـ buffer.

توضح القائمة (١٠-٥) كيفية استخدام الـ GetTextLen والـ GetTextBuf. يحدد الـ procedure أولاً طول الـ buffer الخاص بالـ Memo1. إذا كان هذا الطول اكبر من صفر، يستدعى الـ code الـ StrAlloc، الذى يخصص Len+ بايت واحدة ويعين عنوان الذاكرة بـ P. ينسخ الـ GetTextBuf نص الـ Memo1 الى الذاكرة المخصصة، تأتى بعدها الـ for-loop -تبدل كل حرف فى الـ buffer بالـ FF العشرية. ينسخ الـ SetTextBuf الـ Memo مرة اخرى فى الـ object. وأخيراً، يقوم الـ StrDispose بحذف الذاكرة

## الباب العاشر : العمل مع نص متعدد السطور

المخصصة. جرب هذا ال procedure واضغط الزر لربط النص من والى ال form. ينسخ ال procedure نص ال Memo1 الى حازر ديناميكي، يبدل كل حرف يتبع لعملية ل exclusive-or، ثم ينسخ ال buffer مرة اخرى الى ال Memo1.

القائمة (١٠-٥): استخدمOnClick للتعامل مع memo object text

```
procedure TForm1.Button3Click(Sender: TObject);
var
  I, Len: Integer;
  P: PChar;
begin
  Len := Memo1.GetTextLen;
  if Len = 0 then
    ShowMessage('Text buffer is empty!')
  else begin
    P := StrAlloc(Len + 1);
    try
      Memo1.GetTextBuf(P, Len);
      for I := 0 to Len do
        P[I] := Chr(Ord(P[I]) xor $ff);
      Memo1.SetTextBuf(P);
    finally
      StrDispose(P);
    end;
  end;
end;
```

توضح القائمة جانباً من العمل مع ال PChar pointers. بالرغم من ان متغير PChar يعتبر pointer الى ال Char byte، يمكنك استخدامه وفكر به كأنه array. على سبيل المثال، تعين هذه العبارة الحرف A للرمز الأول string buffer المسمى بـ P:

```
P[0] := 'A';
```

فى القائمة (١٠-٥) يمكنك استبدال ال GetTextBuf بال GetSelTextBuf لنسخ الخاص بك. فيما عدا هذا تعتبر ال functions شئ

واحداً. (إذا جربت هذا، اختر كل النص كل مرة قبل ضغط الزر، أو يكون النص لم يتم التعامل بشكل سليم).

### فهم الـ Tab والـ Enter:

اعتماداً على احتياجات واجهة التطبيق الخاصة بك للـ Memo، فإن ضغط الـ Enter و الـ Tab يمكن ان يكون له تأثيرات متنوعة. على سبيل المثال، قد تريد ان يقوم المستخدمون بإدخال سطور متعددة في نافذة الـ Memo - في هذه الحالة، فإنك في الغالب تريد ضغط الـ Enter لتبدأ سطر جديد. اذا لم تكن تريد ان يعمل الـ Enter بهذه الطريقة، يمكنك برمجة المفتاح على ان يحول الـ focus الى الـ control اخر. يمكنك ايضاً تحديد ما اذا كان ضغط مفتاح الـ Tab يؤدي إلى إضافة الـ tab control code في نص الـ Memo، أو يحول الى الـ control التالي في ترتيب الـ Tab.

حدد خاصية الـ WantReturns بـ True لإدخال سطور جديدة في نص الـ Memo عندما يضغط المستخدمون الـ Enter. حدد خاصية الـ WantReturns بـ False لتوجيه ضغط الـ enter الى الـ form. ولكن، هذا التحديد يتطلب تحديداً مناسباً لخاصية الـ KeyPreview للـ form. وعادة، اذا كانت الـ WantReturns محددة بـ True، يجب ان تكون الـ KeyPreview محددة بـ False حتى يتلقى الـ control، وليس الـ form ضغط مفتاح الـ Enter. اذا كان الـ WantReturns محددة بـ False، يجب ان تكون الـ KeyPreview محددة بـ True؛ والا يستمر الـ form في حالة عدم تلقى لضغط مفتاح الـ Enter في الـ OnKeypress الخاص به.

حدد الـ WantReturns بـ True لإدخال tab control codes في نص الـ Memo عندما يضغط المستخدمون الـ Tab. حدد الـ WantReturns بـ False، لنقل الـ focus إلى control آخر عند تلقى ضغط مفتاح الـ Tab. عندما تكون الـ WantReturns محددة بـ True، يمكنك فحص الـ Memo بضغط الـ Tab، ولكن لا يمكنك ضغط الـ Tab لتخرج.

ان ضغط الـ Ctrl+Enter يبدأ دائماً سطر جديد في الـ Memo، بغض النظر عن قيمة الـ WantReturns. وكذلك، فإن ضغط الـ Ctrl+Tab تضيف دائماً tab control code بغض النظر عن قيمة الـ WantTabs.

بارسال رسالة Windows الى ال Memo ، يمكنك تغيير مسافات tab controls . توضح ال code التالية هذه التقنية ويمكنك استخدامها في OnCreate الخاص بال form . ان ال Tabstops typed-constant array له قيمة Integers واحدة فقط - في هذه الحالة ، محددة بـ ١٨ . ولان اغلب ال fonts نسبية ، فإن ال Windows يقيس عدد مرات توقف ال Tab في dialog box units بدلاً من الحروف ، (ان تحديد ال Tab الافتراضى هو ٣٢ و dialog units) . ويستدعى ال OnCreate ال Perform method (كل ال components لديها هذا ال methods) لإرسال رسالة ال em\_SetTabStops لل Memo . ان ال parameter الواحد هو الذى يحدد لل TabStops مدخل واحد ، يتكرر لكل tabs . انظر ال em\_SetTabStops فى ملف ال Win32.hlp لمزيد من المعلومات حول تحديد مواصفات ال tabs فى . text controls .

وال code التالى ايضاً يوضح كيفية تمرير متغير الى ال LParam parameter الخاص بالرسالة وذلك بطريقة . ال type-casting بالتحويل الى قيمة ال Longint . فى هذا المثال ، تحدد علامة ال @ عنوان ال TabStops array . والتعبير التام Longint(@TabStops) يمرر عنوان ال array كقيمة Longint ذات ٣٢ بت :

```
procedure TForm1.FormCreate(Sender: TObject);
const
  TabStops: array[0 .. 0] of Integer = (18);
begin
  Memo1.Perform(em_SetTabStops, 1, Longint(@TabStops));
end;
```

لتمرير أكثر من tab stop واحدة ، قم بتغيير تعريف الثابت فى ال code السابقة لشيء مثل :

```
const
  TabStops: array[0 .. 3] of Integer = (4, 8, 12, 16);
```

وكذلك قم بتغيير الرقم واحدة الى رقم اربع فى استدعاء ال Memo1.Perform . اذا كنت تمرر tab stop واحدة فقط كما فى القائمة ، لا



يجب عليك إنشاء array . يمكنك بدلاً من ذلك تعريف الـ TabStops على أنه عدد صحيح (والتي، بعد كل شيء، هي نفسها كـ array التي لها قيمة واحدة .

## الـ Clipboard و Text:

بالرغم من نفعيتها المحدودة، تعتبر الـ Clipboard الخاصة بالـ Windows محطة مشهورة لبث المعلومات من وإلى التطبيق . إن غالبية مستخدمي الـ Windows يتعلمون بسرعة كيف يقصون وينسخون النص ، بالرغم من أنهم قد لا يدركون أنهم يستخدمون الـ Clipboard للقيام بهذه الأعمال . ولأن الـ Edit components ، الـ MaskEdit ، والـ Memo تدعم تلقائياً القص ، والنسخ ، واللصق ، فإن استخدام هذه الـ objects هو كل ما تحتاجه لتوفر خدمات الـ clipboard في تطبيقك .

**ملحوظة:** والـ Database components مثل الـ TDBEdit ، TDBMaskEdit ، TDBImage ، والـ TDBMemo أيضاً تدعم الـ clipboard methods الموضحة هنا .

Note

لكي تجعل الـ clipboard أسهل استخداماً ، وخاصة للمبتدئين الذين لم يتمكنوا من مهارات أوامر لوحة مفاتيح الـ Windows ، استدع الـ methods الـ CopyToClipboard ، الـ CutToClipboard ، والـ PasteFromClipboard في الـ Edit ، الـ MaskEdit ، والـ Memo . على سبيل المثال ، لقص النص المختار من الـ Memo إلى الـ clipboard ، استخدم هذه العبارة ، ربما في الـ OnClick الخاص بالـ Button ، أو في الـ procedure الخاص بأمر الـ Edit/Cut الخاص بالقائمة :

```
Memo1.CutToClipboard;
```

وتعمل الـ procedure الأخرى بطريقة مشابهة ولا تتطلب أية parameters . والطريق إلى النجاح هو أن تدرك أن الـ CopyToClipboard والـ CutToClipboard لا يعملان إلا على النص المختار فقط . لنسخ كل نص الـ Memo على الـ clipboard ، استدع الـ Selectall قبل تنفيذ هذه العملية :

```
Memo1.SelectAll;
```

```
Memo1.CopyToClipboard;
```

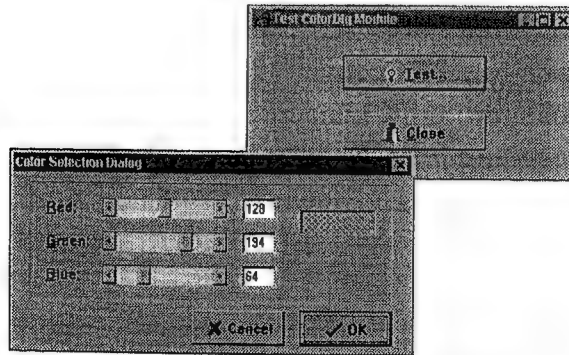
## الباب العاشر : العمل مع نص متعدد السطور

توفر العديد من ال components إمكانات ال scrolling تلقائياً ولا تتطلب أية برمجة . على سبيل المثال ، ال Memo يعرض ، scroll bar رأسية وافقية على حسب الحاجة لتستطيع الوصول الى النص . ويعمل ال Scrolling كما هو متوقع بالضبط .

بالنسبة لمهام ال scroll الأخرى ، لديك خياران . يمكنك إضافة ScrollBar على form أو يمكنك استخدام ScrollBar لتوفير مسطح قابل لل scrolling يمكن أن يحتوي على objects أخرى . يوضح الفصلان التاليان كيفية ال Scrolling في مع هذين ال components الهامتين النافعتين .

### التحرك بال ScrollBar:

**على القرص المدمج:** يقوم ال ScrollBar Component بإنشاء scrollBar object مستقل يمكنك إضافته الى form أو أية حاوية أخرى . بكتابة event handler واحد ، يمكنك إنشاء code تستجيب للتغيرات في موضع ال ScrollBar . يمكن أن يكون لديك أى عدد تحتاجه من ال ScrollBar - فهي وسائل بيئة جيدة في اختيار قيم فيما بين مدى منخفض أو مرتفع . كمثال على هذا المفهوم ، قم بتجربة تطبيق ال Test في دليل ال Source\ColorDlg على القرص المدمج . قم بتشغيل تطبيق ال Test في دليل ال ColorDlg ، واضغط زر ال Test لعرض dialog اختيار اللون الموضح هنا . لاختيار قيم الأزرق ، والأخضر ، والأحمر للألوان الموضحة في شكل (١٠-٣) ال source code لتطبيق ال Test موضحة في القائمة (١٠-٦) .



شكل (١٠-٣): ال Color Selection Dialog يظهر كيفية استخدام ال ScrollBar

قائمة (١٠-٦) \ColorDlg\ColorDlg.pas

unit Colordlg;

interface

uses

Windows, SysUtils, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;

type

TColorDlgForm = class(TForm)

RedSB: TScrollBar;

GreenSB: TScrollBar;

BlueSB: TScrollBar;

RedLabel: TLabel;

GreenLabel: TLabel;

BlueLabel: TLabel;

RedEdit: TEdit;

GreenEdit: TEdit;

BlueEdit: TEdit;

ColorEdit: TEdit;

OkBitBtn: TBitBtn;

CancelBitBtn: TBitBtn;

Bevel1: TBevel;

procedure FormCreate(Sender: TObject);

procedure SBChange(Sender: TObject);

procedure EditChange(Sender: TObject);

procedure FormActivate(Sender: TObject);

procedure OkBitBtnClick(Sender: TObject);

procedure CancelBitBtnClick(Sender: TObject);

private

RedPos, GreenPos, BluePos: Integer; { For undo }

EditControls: array[0 .. 2] of TEdit;

```
ScrollBars: array[0 .. 2] of TScrollBar;
    procedure UpdateColor;
    public
    ColorResult: TColor; { Selected color }
    end;

var
    ColorDlgForm: TColorDlgForm;

implementation

{$R *.DFM}

{- Update ColorResult using scrollbar positions }
procedure TColorDlgForm.UpdateColor;
begin
    ColorResult := RGB(
        RedSB.Position, GreenSB.Position, BlueSB.Position);
    ColorEdit.Color := ColorResult; { Show color }
end;

{- Initialize TObject control arrays }
procedure TColorDlgForm.FormCreate(Sender: TObject);
begin
    EditControls[0] := RedEdit;
    EditControls[1] := GreenEdit;
    EditControls[2] := BlueEdit;
    ScrollBars[0] := RedSB;
    ScrollBars[1] := GreenSB;
    ScrollBars[2] := BlueSB;
end;

{- Update values in Edit boxes for ScrollBar changes }
```

```

procedure TColorDlgForm.SBChange(Sender: TObject);
begin
  with Sender as TScrollBar do
    EditControls[Tag].Text := IntToStr(Position);
    UpdateColor;
  end;

```

```

{- Update scrollbar positions for Edit box changes }
procedure TColorDlgForm.EditChange(Sender: TObject);
begin
  with Sender as TEdit do
    ScrollBars[Tag].Position := StrToInt(Text);
  end;

```

```

{- Save scrollbar positions for possible undo }
procedure TColorDlgForm.FormActivate(Sender: TObject);
begin
  RedPos := RedSB.Position;
  GreenPos := GreenSB.Position;
  BluePos := BlueSB.Position;
end;

```

```

{- Respond to OK button. Accept changes. }
procedure TColorDlgForm.OkBitBtnClick(Sender: TObject);
begin
  ModalResult := mrOk; { Close Window. Color in ColorResult. }
end;

```

```

{- Respond to Cancel button. Undo changes. }
procedure TColorDlgForm.CancelBitBtnClick(Sender: TObject);
begin
  RedSB.Position := RedPos;
  GreenSB.Position := GreenPos;

```

## الباب العاشر : العمل مع نص متعدد السطور

```
BlueSB.Position := BluePos;
ModalResult := mrCancel;
end;

end.
```

يذكر الجدول (١٠-٢) بعض الخصائص الرائعة للـ ColorDlg components، والتي توضح بعض التقنيات البيئية الممتعة. تحدد الـ labels الثلاث الـ Alt accelerator في خصائص الـ Caption التابعة لهم، وتحدد قيم الـ FocusControl لهم للـ Edit المناسب. على سبيل المثال، قم بتشغيل برنامج الـ Test واضغط Alt+G لاختيار الـ GreenEdit.

إن كلاً من الـ ScrollBars الثلاثة في النافذة لها غالباً قيمة LargeChange. هذا يؤثر على كم الـ scrolling عندما يضغط المستخدمون داخل الـ bar- في أغلب الحالات، يجب أن تحدد الـ LargeChange أكبر من الـ SmallChange (التي تساوى واحد عادة)، والذي يؤثر على كم الـ scrolling لأزرار الأسهم للـ ScrollBars وتحدد الـ ColorDlg ScrollBars قيم الـ Max. الخاصة بهم أيضاً بـ ٢٥٥. تمثل نوع بيانات الـ TColor الخاصة بـ Delphi الألوان لمجموعة من الأحمر والأخضر والأزرق (RGB)، كلاً منها في المدى من صفر إلى ٢٥٥.

جدول (١٠-٢): خصائص الـ ColorDlg

Component	Name	Property	Value
Form	ColorDlgForm	Caption	Color Selection Dialog
Label	RedLabel	Caption	&Red:
		FocusControl	RedEdit
Label	GreenLabel	Caption	&Green:
		FocusControl	GreenEdit
Label	BlueLabel	Caption	&Blue:
		FocusControl	BlueEdit
ScrollBar	RedSB	LargeChange	10
		Max	255
		Tag	0

ScrollBar	GreenSB	Tag	1
ScrollBar	BlueSB	Tag	2
Edit	RedEdit	Tag	0
Edit	GreenEdit	Tag	1
Edit	BlueEdit	Tag	2

لاحظ أيضاً فى جدول (١٠-٢) أن الـ Edit والـ ScrollBar تحدد قيم الـ Tag الخاصة بها بصفر، وواحد، وإثنين. يستخدم البرنامج هذه القيم لربط الـ components فى وقت التشغيل. لترى كيف يعمل، اختر الـ TColorDlgForm class فى القائمة (١٠-٦). يعرف الـ private section الـ arrays من نوع الـ objects:

EditControls: array[0 .. 2] of TEdit;

ScrollBars: array[0 .. 2] of TScrollBar;

توفر الـ arrays وسائل يسيرة للوصول الى كلاً من الـ forms أو الـ Edit والـ ScrollBar. تذكر أن متغيرات الـ TEdit والـ TScrollBar تعد references. لذا فالـ arrays يحتويان على pointers الى الـ objects، وليست الـ objects ذاتها. لبدء الـ array، يقوم الـ OnCreate الخاص بالـ form بتعيين الـ objects لمواقع الـ array، تساوى قيم الـ Tag Object. على سبيل المثال، تعين هذه العبارة الجزء الأول فى الـ EditControls لتشير الى الـ RedEdit:

EditControls[0] := RedEdit;

بعد إعداد الـ component reference arrays، يمكن للبرنامج أن يستخدم قيم الـ Tag كـ index للوصول الى الـ objects الفردية. على سبيل المثال، اختبر الـ SBChange procedure، الذى يتعامل مع الـ OnChange event للـ ScrollBar. إن كل الـ OnChange خاص بـ ScrollBar محدد بنفس الـ event handler. فى الـ procedure، يعتبر الـ Sender parameter الذى تم تمريره للـ procedure هو ScrollBar الذى قد تغيره. لتحديث الـ Edit المرتبط به بحيث تتماشى قيمة العدد الصحيح الخاص به مع موضع الـ ScrollBar، يستخدم الـ procedure عبارة with ليعامل الـ Sender على أنه TScrollBar object، ثم ينفذ هذه العبارة:

```
EditControls[Tag].Text := IntToStr(Position);
```

يقوم الـ `IntToStr` بتحويل قيمة خاصية الـ `ScrollBar Position` الرقمية إلى `string`، ويعين الناتج لخاصية الـ `Text` التابعة للـ `Edit control`، باستخدام الـ `EditControls array` والـ `Tag index` للإشارة إلى الـ `Edit` المرتبط.

يؤدي الـ `EditChange procedure` مهمة شبيهة. عندما تدخل قيمة جديدة في نافذة الـ `Edit`، فإن الـ `OnChange` الناتج يقوم باستدعاء الـ `EditChange`. يستخدم الـ `procedure` عبارة الـ `with` لمعامل الـ `Sender` على أنه `TEdit`، ثم ينفذ هذه العبارة:

```
ScrollBars[Tag].Position := StrToInt(Text);
```

تقوم الـ `StrToInt Function` بتحويل خاصية الـ `Text` للـ `Edit` إلى قيمة عدد صحيح، والتي تحددها مرة أخرى العبارة إلى `ScrollBar Position`. هذا يجعل الـ `thumb box` في `ScrollBar` ينتقل إلى موضع يتلائم مع الـ `Edit`. إن الـ `ScrollBar array` وخاصية الـ `Tag` يجعلان من السهل اختيار الـ `ScrollBar` المرتبط بالـ `Edit` الذي ولد الـ `OnChange`.

إن الـ `EditChange` و `SBChange procedures` يتعاملان مع جميع الـ `OnChange event` للـ `Edit` والـ `ScrollBar` الستة الخاصة بالـ `dialog`. يمكن للـ `procedures` أن يتعاملتا مع أي عدد من الـ `objects` المرتبطة، ويعرضا تقنية مفيدة في تقليل حجم الـ `code`. عندما يكون لديك `component object` عديدة مرتبطة بالنافذة، ففكر في استخدام `array` و `Tag index` للإشارة إليهم في الـ `event handler` المشتركة.

يوضح الـ `Procedure UpdateColor` كيفية استخدام قيم الـ `ScrollBar Position`. في هذه الحالة، ترجع الـ `RGB function` قيمة `LongInt` بإعطائها ثلاثة بايت جزئية تمثل ألوان الأزرق والأخضر والأحمر. يحدد البرنامج اللون الناتج إلى متغير الـ `ColorResult`، المعروف في الـ `public section` للـ `TColorDlgForm class`.

يعرض الـ `UpdateColor` أيضاً اختيار اللون التالي باستخدام خدعة بسيطة أسرع من الأسلوب الجرافيكي الواضح. في إحدى نسخ البرنامج، قمت باستخدام الـ `Shape` لعرض اللون، ولكن بسبب الطريقة التي يعمل بها الـ `Windows`، يؤدي كل تغيير باللون إلى أن يسمح الـ `Shape` نفسه ليصبح ذا لون أبيض، مما قد



ينتج وميضاً مزعجاً. (ولكن، اعتماداً على نظامك، قد لا ترى هذا). لحل المشكلة، قمت باستخدام الـ Edit، حددت الـ Enabled والـ TabStop بـ False، وحددت الـ ReadOnly بـ True. لا يمكنك اختيار أو إدخال نص في الـ Edit الناتج. يقوم الـ UpdateColor Procedure بتعيين الـ ColorResult الى خاصية الـ Color بالـ Edit، والذي لا تعيد رسم نفسها بين عمليات التحديث. عندما تحتاج مربع ملون بسيط، جرب الـ Edit بدلاً من الـ Shape.

يقوم الـ ColorDlg بتنفيذ ميزة الـ undo، والتي تحتاجها جميع الـ dialogs التي لديها زر Cancel في هذه الحالة، يقوم الـ OnActivate الخاص بالـ form بحفظ قيمة الـ Position للـ ScrollBar الحالي في ثلاث متغيرات الـ RedPos، الـ GreenPos، والـ BluePos. اذا ضغط المستخدم الـ Close لإنهاء الـ dialog، يقوم الـ OnClick الخاص بالزر بتعيين القيمة التي حفظها الى خصائص الـ Position للـ ScrollBar مرة اخرى.

للوهلة الأولى، قد لا تبدو هذه التعينات كافية لإلغاء كل تغييرات الـ dialog تماماً. والذي لا يبدو واضحة هو ان تعيين قيم جديدة للـ ScrollBar يولد الـ OnClick events. لذلك، فإن ثلاث تعينات في الـ CancelBtnClick تؤدي الى ثلاث استدعاءات للـ SBChange، والذي يعيد تحديد الـ Edit controls ويقوم بتحديث نافذة اللون. تتبع هذه الـ events يضغط F7 و F8 لتشغيل برنامج الـ Test، أو حدد نقاط إنطلاق في مواضع استراتيجية، وتحقق من التابع الذي يحدث عندما تضغط الـ Cancel.

لاستخدام dialog اختيار اللون، اصف الـ ColorDlg.pas الى أى مشروع وأضف الـ ColorDlg للـ uses directive للـ form الرئيسية. استدع الـ ShowModal function لعرض نافذة الـ dialog:

`ColorDlgForm.ShowModal;`

يرجع الـ ShowModal قيمة تشير الى أى زر تم ضغطه لإنهاء الـ dialog. يمكنك فحص هذه القيمة واتخاذ التصرف المناسب. على سبيل المثال، بدلاً من إستدعاء الـ ShowModal كما في السطر السابق، استخدم هذه الـ code المعقدة بعض الشيء:

```
with ColorDlgForm do
if ShowModal = mrOk then
  ShowMessage(Format('Color value = $%-6x', [ColorResult]));
```

إذا أرجع الـ ShowModal الـ mrOk، يعرض الـ ShowModal قيمة الـ ColorResult من الـ ColorDlgForm، باستخدام الـ Format لإدخال القيمة بالكسر العشري في الـ dialog الرسالة. ويقوم النص ذو الشكل المضحك، وهو، "%-6x"، بتهيئة الـ ColorResult على أنه قيمة عشرية (x) ذات مسافة (-) في ستة أعمدة ذات أصفار من ناحية اليسار (6).

إن الـ forms والـ components الأخرى تستخدم الـ TScrollBar class كخاصية للـ scroll bars الرأسى والأفقى. إن هذه الـ class لها نفس خصائص والـ methods كما للـ TScrollBar class، التى يمكنك من استخدامها كـ objects مستقلة على الـ form.

### التحريك بالـ ScrollBox

إن الـ ScrollBox يشبه الـ Panel التى يمكن أن تتحرك إلى أعلى، وأسفل، يساراً ويميناً. يعتبر الـ ScrollBox حاوية يمكن أن تحمل الـ objects أخرى الأزرار، الـ labels، الـ Edit، الـ radio buttons - وحتى الـ ScrollBox panels الأخرى.

إن إنشاء الـ ScrollBox يعتبر أمر سهلاً فقط أضف واحداً على الـ form، وحدد حجمة أو حدد الـ Align بـ alClient لتملأ الـ panel الـ client area الخاصة بالنافذة. ويعتبر هذا التحديد جيد لإنشاء الـ dialog boxes قابلة للتحريك و الـ forms لإدخال البيانات ذات الـ controls كثيرة جداً بحيث تتناسب بشكل مريح داخل نافذة واحدة. تحدث الـ Scrolling بصورة تلقائية، وتظهر الـ scroll bars حسب الحاجة لتمكن المستخدمين من الوصول إلى الـ controls الواقعة خلف حدود النافذة.

**فكرة:** وهناك طريقة أخرى للـ form، وهى بأن تحدد خاصية الـ AutoScroll لها بـ True. ولكن لا تزال الـ ScrollBox مفيدة في ظروف عديدة، كما يوضح هذا الفصل.

قم بتصميم ال ScrollBox الخاصة بك بالسعة الكاملة لها . على سبيل المثال ، يمكنك تكبير نافذة ال form ، وإضافة كل ال controls فى المكان الذى تريده فيه . بعد تصميم النافذة ، قم بتقليص ال form المصممة لترجع الى حجمها النهائى مرة اخرى .

وتعتبر ال ScrollBoxes مفيدة ايضاً فى حالة اختلاف قياس ال form فى حالات ال Resolution المختلفة . على سبيل المثال ، يمكنك تصميم full screen entry form بمقياس ١٠٢٤×٦٧٨ التى لازالت تعمل بشكل صحيح على الحاسبات العملية بقياس ٦٤٠×٤٨٠ . على الحاسب المعملى ، يجب على المستخدم تحريك النافذة ليصل الى كل ال controls .

ان ال controls التى يمكن ان تتلقى input focus (مثل edit controls) تتحرك تلقائياً تجاه الرؤية . حاول إضافة إثنى عشرة Edits على ال ScrollBox . قم بتشغيل البرنامج واضغط Tab . عندما يتغير input focus يتحرك ال Edit الحالى تلقائياً الى اتجاه الرؤية .

فى بعض الأحيان ، قد تحتاج الى عرض controls معين . على سبيل المثال ، قد يختار المستخدم أمر برنامج أو يضغط زرأ . يمكنك استدعاء ال SetFocus method الخاص بال object ، أو يمكنك استدعاء الخاص بال ScrollInView . procedure . على سبيل المثال ، لضمان ال Edit7 مرثياً ، قم بتنفيذ العبارة التالية :

ScrollBox1.ScrollInView(Edit7);

### components و controls :

عندما نتعرف اكثر على حاويات مثل ال ScrollBox ، سوف تكتشف الحاجة الى التوصل الى ال components بطرق متنوعة . بالرغم من ان كل component objects تعتبر متوفرة بالاسم ، مثل ال Label2 أو ال Edit9 ، وليس من الكافى دائماً الإشارة الى ال objects منفردة . فى form أو ScrollBox (أو Panel) ذات عشرات الازرار ، وليس من المستحب ان تكتب code مثل التالية :

Button1.Enabled := False;

Button2.Enabled := False;

...

Button38.Enabled := False;

## الباب العاشر : العمل مع نص متعدد السطور

ان الحاويات كـ form والـ ScrollBox تقدم الـ Components array ،  
الذي يمكنك استخدامه للوصول الى كل الـ objects المملوكة للـ object الرئيسى .  
الـ ComponentCount يساوى عدد تلك الـ components . يمكنك استخدام  
هذه الخصائص لتكتب loops مثل هذه :

```
for I := 0 to ComponentCount - 1 do
```

```
  Components[I].Enabled := False;
```

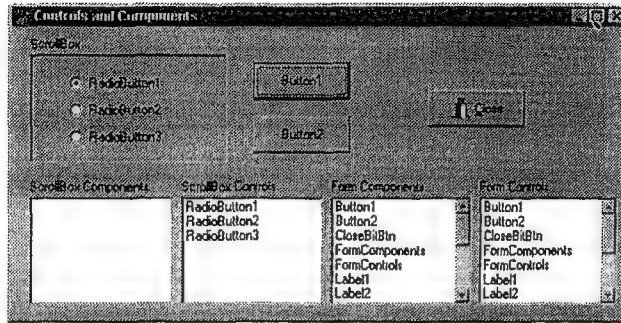
توفر الحاويات ايضاً خصائص مشابهة للـ array والـ count والـ Controls  
والـ ControlIndex . ولكن هذه الخصائص تساعدك للوصول الى الـ  
components التى تعتبر ايضاً نوافذ صغيرة . توفر خصائص الـ Controls والـ  
ControlIndex وصلاً الى component objects . ويذكر والـ Controls  
array علاقات النافذة مع component objects . والـ Controls array تحتوى  
على قائمة بعلامه الـ object . ولا يعتبر الـ arrays واحداً دائماً . على سبيل المثال ،  
يملك الـ ScrollBox للـ components الخاصة به على انها child controls  
window . والـ form التى تملك الـ ScrollBox تملك نفس تلك الـ objects  
components .

**على القرص المدمج:** هذه المفاهيم ليست اكااديمية خالصة - يمكنك

استخدامها تبعاً لمصالحك فى كتابة code تؤثر على مجموعات من الـ  
objects . لتفهم جيداً الفرق بين الـ Controls array والـ



components ، قم بتشغيل تطبيق الـ ContComp ( Controls و  
Components ) الموجود على القرص المدمج فى دليل الـ Source\ContComp .  
يذكر البرنامج الـ components والـ controls فى اربعة قوائم للـ ScrollBox وللـ  
form الرئيسيه . يوضح شكل ( ١٠-٤ ) البرنامج . تذكر القائمة ( ١٠-٧ ) للـ  
source code ، التى تتضمن procedures التى يمكنك استخدامها لفحص  
علاقات الـ components و form control .



شکل (۱۰-۴): يعرض الـ ContComp العلاقات الـ components والـ controls والـ form objects مع ScrollBox، الذي يحتوى على ثلاث RadioButton controls، والـ form

القائمة (۱۰-۷): ContComp/Main.pas

unit Main;

interface

uses

Windows, SysUtils, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, StdCtrls, Buttons;

type

```
TMainForm = class(TForm)
  ScrollBox1: TScrollBox;
  Label1: TLabel;
  RadioButton1: TRadioButton;
  RadioButton2: TRadioButton;
  RadioButton3: TRadioButton;
  Button1: TButton;
  Button2: TButton;
  ScrollBoxComponents: TListBox;
  ScrollBoxControls: TListBox;
  FormComponents: TListBox;
  FormControls: TListBox;
```

```
Label2: TLabel;  
Label3: TLabel;  
Label4: TLabel;  
Label5: TLabel;  
CloseBitBtn: TBitBtn;  
procedure FormCreate(Sender: TObject);  
private  
{ Private declarations }  
public  
{ Public declarations }  
end;
```

```
var  
MainForm: TMainForm;
```

implementation

```
{ $R *.DFM }
```

```
procedure ListControls(ListBox: TListBox;  
Control: TWinControl);
```

```
var  
I: Integer;  
begin  
with Control do  
for I := 0 to ControlCount - 1 do  
ListBox.Items.Add(Controls[I].Name);  
end;
```

```
procedure ListComponents(ListBox: TListBox;  
Component: TComponent);
```

```
var  
I: Integer;
```

```

begin
  with Component do
    for I := 0 to ComponentCount - 1 do
      ListBox.Items.Add(Components[I].Name);
    end;

  procedure TMainForm.FormCreate(Sender: TObject);
  begin
    ListControls(ScrollBarControls, ScrollBox1);
    ListComponents(ScrollBarComponents, ScrollBox1);
    ListControls(FormControls, MainForm);
    ListComponents(FormComponents, MainForm);
  end;
end.

```

لفحص علاقات ال control و ال components فى تطبيقاتك، إنسخ ال procedure ListComponents و ال ListControls فى أى unit module. لن تريد فى الغالب ان تتضمن ال code فى التطبيق الأخير، لذا لم اجعلها جزءاً من class. فى أى event handler (يستخدم البرنامج ال OnCreate الخاص بال form) استدع كلا ال (procedure) لإدخال ال components و ال controls فى ListBoxes. قم بتمرير ال ListBox و ال component objects على انهما arguments قد تقوم بتمرير أية object منحدره من ال TComponent الى ال ListComponents و أية object منحدره من ال TWinControl الى ال ListControls. ولكن، و object components يجب ان توفر ال Control و ال Components arrays.

ونعود للـ ScrollBoxes، كما يوضح شكل (١٠-٤)، ان ال ScrollBoxes لا يملك ثلاث objects component RadioButton. ولكن، ال objects تعتبر Child ScrollBox. من هذه المعلومة، يمكنك كتابة procedure فعلاً لأداء عمليات على جميع ال ScrollBox control. على سبيل المثال، جرب ال OnClick الموجود فى القائمة (١٠-٨) للـ Button الثالث فى برنامج ال ContComp لإبطال ال RadioButtons الخاصة بال ScrollBox. يتوصل ال code لكل Button من خلال ال ScrollBox control.

إرجع مرة أخرى الى شكل (١٠-٤) "أوقم بتشغيل برنامج الـ ContComp واختبر قوائم الـ FormComponents والـ FormControls. تمتلك الـ form كل الـ objects الموجودة في البرنامج، بالرغم من ان الـ RadioButtons الثلاثة داخل الـ ScrollBox. ولكن، الـ control مثل الـ Child Window يمكن ان يكون له أب واحد فقط، بذلك، فإن الـ Controls array الخاص بالـ form لا يذكر الـ RadioButtons.

#### القائمة (١٠-٨): OnClick، لإضافة زر

```
procedure TMainForm.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  with ScrollBox1 do
    for I := 0 to ControlCount - 1 do
      Controls[I].Enabled := False;
end;
```

تساوى قيمة خاصية الـ ComponentIndex مع قيمة الـ object index في الـ Components array التابعة لـ object معين. بنفس الطريقة، تساوى الـ Control Index الـ object index في الـ Components array التابع لـ object معين. اذا كانت الـ T object Label مملوكة للـ forms، واذا كانت الـ T.ComponentIndex تساوى K إذن الـ Form.Components[K] تشير الى الـ T. بنفس الطريقة، اذا كانت الـ T control وهو Child Window من الـ form، فالـ Form.Controls[K] تشير الى الـ T. اذا قضيت بعض الوقت في فحص هذه العلاقات، سوف تكتشف العديد من الفرص للوصول الى الـ object بطريقة فعالة من خلال الـ Controls array والـ Components.

#### الـ StringGrids:

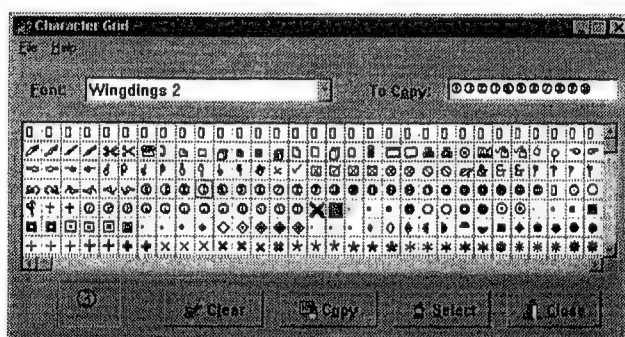
على القرص المدمج: سوف نرى الآن مثلاً يعد واحداً من أكثر الـ Delphi component تقبلاً، الا وهو الـ StringGrid. (قدم الباب السابق هذا component المفيد). ان الـ StringGrid، الذى يعد دفتر





حسابات (أو spreadsheet) تام، ينظم الـ string و object data الأخرى فى تخطيط جدول من صفوف واعمدة. يوضح برنامج الـ CharGrid على القرص المدمج فى دليل Source\CharGrid كيفية استخدام الـ StringGrid. كما هو موضح فى شكل (١٠-٥)، يعرض البرنامج StringGrid يوضح كل الرموز فى أى Windows font. اختر font من الـ Font ComboBox. اضغط مرتين أى رمز فى الـ Grid لإضافته الى نافذة الـ To Copy Edit. أو، يمكنك أيضاً ضغط زر الـ Select، إضغط زر الـ Copy لنسخ الحروف المختارة بالـ clipboard. يمكنك عندئذ الانتقال الى تطبيق آخر ولصق الحروف فى أى Text edit. توضح القائمة (١٠-٩) الـ source code للـ CharGrid.

**فكرة:** استخدام الـ StringGrid. لإضافة حروف صعبة الكتابة مثل حروف حق الطبع (C) والعلامة التجارية (R) فى الـ Pascal string.



شكل (١٠-٥): تطبيق الـ StringGrid يوضح كيفية استخدام الـ StringGrid الخاص بـ Delphi

القائمة (١٠-٩): CharGrid\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
Windows, SysUtils, Messages, Classes, Graphics, Controls,  
Forms, Dialogs, Grids, StdCtrls, Buttons, ExtCtrls,
```

Menus,  
About;

type

```
TMainForm = class(TForm)
StringGrid1: TStringGrid;
FontCB: TComboBox;
FontLabel: TLabel;
CloseBitBtn: TBitBtn;
CopyEdit: TEdit;
CopyLabel: TLabel;
SelectBitBtn: TBitBtn;
ClipBitBtn: TBitBtn;
CharLabel: TLabel;
Bevel1: TBevel;
ClearBitBtn: TBitBtn;
MainMenu1: TMainMenu;
FileMenu: TMenuItem;
FileExit: TMenuItem;
HelpMenu: TMenuItem;
HelpAbout: TMenuItem;
procedure FormCreate(Sender: TObject);
procedure FontCBChange(Sender: TObject);
procedure FontCBKeyDown(Sender: TObject; var Key:
Word;
Shift: TShiftState);
    procedure StringGrid1DblClick(Sender: TObject);
procedure StringGrid1KeyDown(Sender: TObject;
var Key: Word; Shift: TShiftState);
    procedure StringGrid1SelectCell(Sender: TObject;
Col, Row: Longint; var CanSelect: Boolean);
    procedure FileExitClick(Sender: TObject);
procedure HelpAboutClick(Sender: TObject);
procedure SelectBitBtnClick(Sender: TObject);
```

```

procedure ClipBitBtnClick(Sender: TObject);
    procedure ClearBitBtnClick(Sender: TObject);
    private
    { Private declarations }
    public
    { Public declarations }
    end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

{ Initialize controls }
procedure TMainForm.FormCreate(Sender: TObject);
var
    Ascii, IRow, ICol: Integer;
begin
    { Initialize FontCB ComboBox with font names }
    FontCB.Items := Screen.Fonts;
    { Show current StringGrid font in FontCB's edit box }
    FontCB.ItemIndex :=
        FontCB.Items.IndexOf(StringGrid1.Font.Name);
    { Insert characters into grid }
    Ascii := 0;
    with StringGrid1 do
        for IRow := 0 to RowCount do
            for ICol := 0 to ColCount do
                begin
                    Cells[ICol, IRow] := Chr(Ascii);
                    Inc(Ascii);
                end;
            end;
        end;
    end;
end;

```

```
end;

{ Assign sample character and font }
with StringGrid1 do
    CharLabel.Caption := Cells[Row, Col];
    CharLabel.Font.Name := StringGrid1.Font.Name;
end;

{ Change grid, edit box, and sample to selected font }
procedure TMainForm.FontCBChange(Sender: TObject);
begin
    StringGrid1.Font.Name := FontCB.Text;
    CopyEdit.Text := ""; { Optional: Erase current entries }
    CopyEdit.Font := StringGrid1.Font;
    CharLabel.Font.Name := StringGrid1.Font.Name;
end;

{ Close FontCB drop-down list on pressing Enter or Esc }
procedure TMainForm.FontCBKeyDown(Sender: TObject;
    var Key: Word; Shift: TShiftState);
begin
    if Key in [vk_Return, vk_Escape] then
        begin
            FontCB.DroppedDown := False;
            Key := 0;
        end;
end;

{ Grid double-click event handler }
procedure TMainForm.StringGrid1DblClick(Sender: TObject);
begin
    with CopyEdit, StringGrid1 do
        Text := Text + Cells[Col, Row];
    end;
```

```

{ Select character on pressing Enter or Space }
procedure TMainForm.StringGrid1KeyDown(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
  if Key in [vk_Return, vk_Space] then
    StringGrid1DbClick(Sender); // Same as double-click grid
end;

{ Show selected character }
procedure TMainForm.StringGrid1SelectCell(Sender: TObject; Col,
  Row: Longint; var CanSelect: Boolean);
begin
  CharLabel.Caption := StringGrid1.Cells[Col, Row];
end;

{ File|Exit menu command }
procedure TMainForm.FileExitClick(Sender: TObject);
begin
  Close;
end;

{ Help|About menu command }
procedure TMainForm.HelpAboutClick(Sender: TObject);
begin
  AboutForm.ShowModal;
end;

{ Select button click handler }
procedure TMainForm.SelectBitBtnClick(Sender: TObject);
begin
  StringGrid1DbClick(Sender); // Same as double-clicking
  grid
end;

{ Copy selected characters to clipboard }

```

```
procedure TMainForm.ClipBitBtnClick(Sender: TObject);
begin
  with CopyEdit do
    begin
      if SelLength = 0 then
        SelectAll;    { Select all text if none selected }
        CopyToClipboard; { Copy selected text to
        clipboard }
      end;
    end;

    { Clear text in copy-to edit box }
  procedure TMainForm.ClearBitBtnClick(Sender: TObject);
  begin
    CopyEdit.Text := "";
  end;

end.
```

يعمل ال StringGrid component مثل ال ListBox الثنائي الابعاد .  
تحديد خاصية ال RowCount عدد الصفوف ، وتحدد خاصية ال ColCount عدد  
الاعمدة . على ايسر مستوى ، يمكنك استخدام هذه القيم للوصول الى بيانات ال  
grid من خلال ال Cells . على سبيل المثال ، يقوم برنامج ال OnCreate لل  
بتعيين كل قيم ال ASCII الممكنة ال StringGrid1 object الخاص بالبرنامج بهذه  
العبارات ( ال Ascii ، IRow ، وال ICol ومتغيرات Integer ) :

```
Ascii := 0;
with StringGrid1 do
  for IRow := 0 to RowCount do
    for ICol := 0 to ColCount do
      begin
        Cells[ICol, IRow] := Chr(Ascii);
        Inc(Ascii);
      end;
    end;
```

يمكنك أيضاً التوصل الى grid strings من خلال ال Rows وال Cols ،  
والتي تعتبر single-dimensional string . كل عنصر من ال Rows وال Cols  
يعتبر TStrings string-list object . لتحميل بيانات صف من ملف قرص ،  
مثلاً ، يمكنك استخدام عبارة مثل :

```
with StringGrid1 do
  Rows[0].LoadFromFile('C:\YourFile.Txt');
```

أو ، يمكنك تحميل صفوف متعددة بـ code مثل :

```
with StringGrid1 do
  for I := 0 to RowCount do
    Rows[I].LoadFromFile('C:\YourFile.Txt');
```

عند عمل إضافات عديدة لصفوف واعمدة ال StringGrid (وبشكل عام  
عند إضافة strings لل TStrings وال TStringList ) ، ضع العمليات ما بين بـ  
BeginUpdate و EndUpdate لإعادة عرض ال object مرة أخرى . على  
سبيل المثال ، قد يكون المثال السابق أبسط عندما يصاغ كما يلي :

```
with StringGrid1 do
  for I := 0 to RowCount do
    with Rows[I] do
      try
        BeginUpdate;
        Rows[I].LoadFromFile('C:\YourFile.Txt');
      finally
        EndUpdate;
      end;
```

من الصواب ان تستخدم ال try-finally block كما هو موضح لتأكد ان ال  
EndUpdate قد تم استدعائه ، حتى اذا حدثت أية exceptions فى العبارتين  
الواقعتين تحت try .

تعتبر أغلب إمكانيات ال StringGrid هى نفسها إمكانيات ال TStrings ،  
والتي اختبرتها بعمق ، لذا فلن أزيد هنا . ولكن ، يعرض البرنامج تقنية مفيدة ، وإن  
كانت ليس لها علاقة ، لقوائم ال ComboBox التى قد تجدها معاونة . فى ال

## الباب العاشر : العمل مع نص متعدد السطور

OnKeyDown الخاص بـ FontCB object ، اذا كان الـ Key Parameter الذى تم تمريره يعتبر vk\_Return أو vk\_Escape ، يحدد الـ procedure خاصية الـ DroppedDown للـ ComboBox بـ False ، ويضغط Enter أو Esc لإغلاق نافذة القائمة . وهذه اللمسة البسيطة تساعد على جعل واجهة تطبيق لوحة المفاتيح الخاصة بالـ CharGrid مألوفاً .

يوضح الـ OnKeyDown الخاص بالـ StringGrid كيفية برمجة مفاتيح الـ StringGrid object . وهنا ، اذا كان الـ Key يساوى vk\_Return أو vk\_Space يستدعى الـ StringGrid1DbClick procedure ، الذى يحاكي الضغط مرتين على الـ grid cell . حتى تعمل هذه التقنية ، يجب أن تقوم بإضافة الـ OnDbClick الخاص بالـ grid .

**فكرة:** بشكل عام ، وجدت أنه من الأسهل برمجة الـ event للفأرة أولاً (مثل الـ OnClick والـ OnDbClick) ، ثم إضافة keyboard operations ، إننى استدعى الـ mouse handlers من الـ OnKeyDown والـ OnKeyPress .



## افكار للمستخدم الخبير

- تعود الـ Windows API ShellExecute على الفور - فهى لا توقف التطبيق الحالى لتشغيل آخر ، كما يعتقد بالخطأ بعض واضعى البرامج . بعد أن يستدعى تطبيق A الـ ShellExecute لتشغيل تطبيق B ، من الأمان أن تغلق تطبيق A بينما يظل B يعمل . إن الـ A والـ B تعتبر عمليات مشتركة مستقلة .

- تقوم الـ StringGrid عادة بتثبيت الصف والعمود الأولين التابعين لها فى مكان لأنك غالباً سوف تعرض headers فى هذه الخلية . اذا لم تكن فى حاجة الى headers (كما فى تطبيق الـ CharGrid) ، حدد الـ FixedCols والـ FixedRows بصفر .

- يتم استدعاء الـ OnActivate event للـ form عندما تتلقى الـ form الـ input focus نتيجة تحول المستخدم مرة أخرى الى الـ form من واحدة أخرى فى



نفس التطبيق. إن ال OnActivate event لا يعتبر مساوياً لرسالة ال wm\_Activate بال Windows.

• إن ال VisibleRowCount وال VisibleColCount التابعان لل StringGrid يشيران الى كم الصفوف والأعمدة غير الثابتة فى ال grid. تشير متغيرات ال Row وال Col الى العملية التى لها input focus.

• للكتابة فى ال cell فى ال StringGrid، حدد الخاصية الفرعية Options.goEditing بـ True. اذا أردت أن تمكن المستخدمين من ضغط ال Tab للانتقال من cell الى أخرى، حدد ال Options.goTab بـ True.

• لإعادة تحديد حجم العمود والصف بضغط وسحب الفأرة، حدد ال Options.goColSizing وال Options.goRowSizing بـ True. لتشغيل ضغط وسحب الأعمدة والصفوف الى مواضع جديدة، حدد ال Options.goRowMoving وال Options.goColMoving بـ True.

• عندما يتم إنشاء component وهو يقوم بإدخال نفسه فى قائمة ال Components الخاصة لل component التابع له. وال Ownership تعنى ان ال child components يتم تدميرها عندما يتم تدمير ماليكها. ويحدد ال parent component ال display context. بدون ال parent component، سوف يصبح ال control غير مرئياً. كل ال components التى تنشئها فى وقت التصميم تابعة لل form.

## المشروعات التى يمكنك تجربتها

(١٠-١): استخدام ال ExecuteFile فى ال FMXUtils unit التابعة لـ Delphi لإنشاء control-panel application والذى يعرض ايقونات التطبيق. يجب ان يكون المستخدمون قادرين على ضغط الايقونة لتشغيل البرامج. يمكنك توفير هذا البرنامج من خلال عدة utilities.

(١٠-٢): اكتب بديلاً للـ Windows Notepad، يكون أكثر ملاءمة لوضع ال source code فى قائمة. يجب ان يستخدم برنامجك

## الباب العاشر : العمل مع نص متعدد السطور

monospace (أو يكون لدية أمر ال Font لاختيار ال fonts . فى اماكن اخرى من هذا الكتاب (الباب الثانى عشر ، مثلاً) ، سوف تعرف المزيد عن اوامر البحث والإبدال وكذلك الطبع ، الذى يمكنك إضافتها الى text editor النهائى الخاص بك . ولكن ، لا يزال هذا المشروع تمريناً مفيداً على استخدام ال Memo .

(٣-١٠) : اكتب code لنسخ النص المختار من والى ال Memos مختلفة .

(٤-١٠) : اكتب برنامجاً يرتب ال strings فى ال Memo . (ملحوظة : يمكنك استخدام ال ListBox الذى يقوم بترتيب البيانات بصورة تلقائية عندما تحدد خاصية ال Sorted له بـ True ، أو يمكنك كتابة code ليطبق ال (Sort algorithm) .

(٥-١٠) : اكتب برنامجاً يقوم بال encrypts و decrypts لـ text files . أضف الحماية بكلمة المرور للبرنامج .

(٦-١٠) : أضف ألوان للاحمر والاخضر والازرق الى ال ColorDlg .  
dialog .

### ملخص:

• ال StringGrid وال Memo توفر multiple-line text-handling . ان ال Memo يبدو وكأنك لديك ال Windows Notepad utility فى صورة component . يعتبر ال StringGrid من الناحية العملية (spreadsheet) ينظم قوائم ال strings فى الاعمدة والصفوف .

• ان نسخة ال Memo ذات ال ١٦ بت يمكن ان تحمل الى ٣٢ K فقط من ال Text . والنسخة الاحدث ذات ال ٣٢ بت ، التى تستخدم نوع بيانات ال Object Pascal String ، يمكن ان تحمل كم غير محدد من النص ، ولكنه محدد من ال Windows بـ ٦٤ K .

• يمكنك الوصول الى ال Memo text على انه data object وحيد وذلك باستخدام خاصية ال Text . للوصول الى ال Memo text على انه individual strings ، استخدم خاصية ال Lines . من الممكن ايضاً ، رغم انه من غير المرغوب

فيه، ان تتوصل الى بيانات الـ Memo كـ Char array وذلك باستدعاء methods مثل الـ GetTextBuf و SetTextBuf .

• الـ Text components، مثل الـ Memo والـ Edit، توفر بصورة تلقائية القص والنسخ واللصق للـ clipboard. ولكن، يمكنك استدعاء الـ CopyToClipboard والـ CutToClipboard والـ PasteFromClipboard للـ Edit، والـ MaskEdit، والـ Memo لأداء إنتقالات الـ clipboard تحت تحكم البرنامج.

• ان الـ WinExec function لازالت متاحة، ولكن غير موصى بها. لتشغيل تطبيقات من خارج برنامجك (أو لفتح ملفات فى تطبيقاتها الافتراضية)، استخدم الـ ShellExecute الخاصة بالـ Windows API. ان اسهل الطرق لفعل هذا فى Delphi هى بإضافة الـ FMXUtils unit (الموجودة فى دليل الـ Project\Add to Demos\Doc\Filemanex التابع لـ Delphi) باستخدام أمر الـ Project\uses FMXUtils لقطاع الـ implementation. وكذلك اصف عبارة ExecuteFile المتوفرة فى الـ FMXUtils. form. يمكنك عندئذ استدعاء الـ ExecuteFile المتوفرة فى الـ FMXUtils. يقوم الـ ExecuteFile باستدعاء الـ ShellExecute الخاص بالـ Windows API.

• استخدم الـ ScrollBox على انه range-selection object. استخدم الـ ScrollBox لتوفير سطح قابل للـ scrollable يشبه الـ Panel يمكن ان يحمل controls اخرى.

• فى container object. مثل الـ form أو الـ ScrollBox، تحتوى قائمة Components array بكل الـ components التابع لهذا الـ object. يحتوى الـ Controls array على كل الـ component التى تعتبر child windows. هذه الـ arrays لا تحتوى نفس الـ component بالضرورة. استخدم البرمجة الموجودة بتطبيق الـ ContComp الخاص بهذا الباب لفحص علاقات الـ controls والـ components بين الـ objects. ان فهم هذه العلاقات يمكن ان يساعدك على كتابة code فعالة- على سبيل المثال، loop توقف عمل كل child controls فى الـ ScrollBox دون التأثير على objects أخرى بالـ form.

## الباب العاشر : العمل مع نص متعدد السطور

• تقدم الـ StringGrid عمود وصف لتخزين stars وبيانات object أخرى . استخدم الـ Cells array الثنائي الأبعاد للتوصل الى Cell منفردة . استخدم الـ Rows arrays والـ Cols الأحادية البعد للوصول الى بيانات الـ grid على أنها قوائم strings .

• إن التعامل مع ملفات القرص ، اسماء الملفات ، والدلائل ، والأمور المتعلقة بها يعتبر جانباً هاماً في أغلب مشاريع تطوير البرمجيات . في الباب القادم ، سوف تعرف المزيد عن directory navigation الـ file components .



# الباب الحادى عشر

## Navigating Directories and Files

محتويات هذا الباب،

• Components

• إنشاء directory dialog

• تطوير directory-based utilities

• ملفات Drag-and-drop

• قراءة وكتابة ملفات الـ .ini

يعتبر الـ directory و file management امراً هاماً فى التطبيقات المؤسسة على ملفات . حتى ابسط utility تحتاج الى قراءة وكتابة مواصفات اختيارية فى ملف البدء (.ini). الخاص بالـ Windows ، وفى الغالب تحتاج البرامج الى تقنية واحدة على الاقل فى التعامل مع الملف والموضحة فى هذا الباب .

يقدم Delphi اربعة components للـ directory والملف والتى يمكنك استخدامها لإنشاء directory-navigation dialogs و File Selection . فى هذا الباب ، تستخدم هذه الـ components لإنشاء File Selection dialog وبرنامجاً يمكن ان يقوم بتشغيل أى برنامج تنفيذى . سوف تعرف ايضاً كيفية استخدام الـ TIniFile object لقراءة وكتابة ملفات الـ .ini . وعلى سبيل المثال التعامل مع ملفات الـ .ini ، يوضح هذا الباب الاعمال الداخلية لبرنامج utility معقد نسبياً ، وهو الـ SysColor ، وهو يمكنه ان يغير الالوان الموجودة على الشاشة مثل

ظلال الازرار و title bars . ان البرنامج يقرأ ويكتب عناصر اللون فى ملف ال .ini الخاص بال Windows .

## : Components

فيما يلى directory File components و Delphi

• **DirectoryListBox**: يعرض ويسمح للمستخدمين بتغيير الدليل الحالى والأدلة الفرعية. فى اغلب الحالات، إنك تربط ال DirectoryListBox بال FileListBox لعرض ملفات عندما يتصفح المستخدمين دليل لقرص. ال Win3.1 : Palette .

• **DirectoryListBox**: يعرض ويسمح باختيار كل drives المتصلة بالنظام. إنك تربط بشكل طبيعى ال DirectoryListBox بال DirectoryListBox متى يتمكن المستخدمون من الانتقال الى drives آخر وتصفح ادلته. Win3.1 : Palette .

• **FileListBox**: يعرض ملفات الدليل الحالى. يمكن ان يعرض ال FileListBox كل الملفات أو تلك التى تتماشى مع واحداً أو أكثر من ال wildcard filters، مثل ال \*.pas وال \*.txt. ، إنك تربط ال FileListBox بال Edit لتزود المستخدمين بوسائل إدخال وتحرير اسماء الملف. Win3.1 : Palette .

• **FilterComboBox**: قوائم ال wildcard filters التى يمكن للمستخدمين الاختيار منها لتحديد انواع الملفات التى يعرضها ال FileListBox المرتبط به. Win3.1 : Palette .

**ملحوظة:** بالرغم من ان ال components السابقة توجد فى دليل ال Win3.1 ، فهى مازالت ذات قيمة فى إنشاء مديرى ملفات مخصصة و controls أخرى لتصفح دليل آخر. تعمل ال components بشكل مثالى مع ال Windows 95, 98, NT.



## : إنشاء Directory Dialog

تقدم components الملف والدليل الاربعة الخاصة بـ Delphi مجموعة ادوات Erector التى يمكنك استخدامها لإنشاء File Selection dialogs. فى

## الباب الحادى عشر : Navigating Directories and Files

هذا الفصل ، ستقابل كل components ، ثم تستخدمهم لإنشاء File dialogs Selection الذى يمكنك وضعه للعمل فى اوامر ال FileOpen ، وال FileSave ، والاوامر الأخرى للتعامل مع الملف فى تطبيقك .

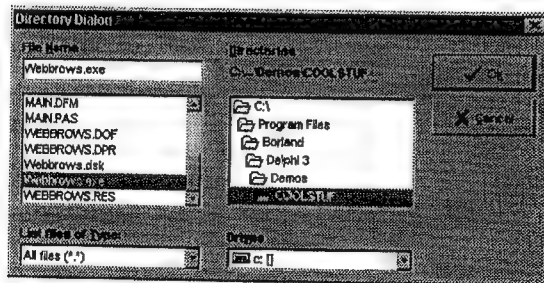
إنك تدرك غالباً أن ال Delphi يقدم ال OpenFileDialog وال SaveDialog على ال Dialogs palette (انظر الباب الثانى عشر) . استخدم التقنيات التالية عندما تحتاج ان تعتاد على ذلك - على سبيل المثال ، لإضافة wildcard filter ، editor ، كواحد من المشروعات المقترحة فى هذا الباب .

### : DirectoryListBox

يوضح شكل (١١-١) ال dialog box الذى تنشئه مؤخراً . فى هذا الفصل . يعرض ال DirectoryListBox تخطيط لشجرة دليل . يمكن للمستخدمين الضغط مرتين واستخدام لوحة المفاتيح لاختيار ادلة فى هذه النافذة .

يمكنك استخدام ال DirectoryListBox بمفرده فى نافذة ، ولكنك عادة تريد عرض ملفات فى الأدلة المختارة . لتفعل هذا ، قم بتعيين اسم ال FileListBox لخاصية ال FileList التابعة لل DirectoryListBox . هذه هى اسهل طريقة ويمكن اداؤها فى نافذة ال Object Inspector ، ولكن يمكنك القيام بالتعيين فى وقت التشغيل كالاتى . أضف ال DirectoryListBox وال FileListBox على ال form ، ثم أدخل هذه العبارة فى ال OnCreate الخاص بال form :

```
DirectoryListBox1.FileList := FileListBox1;
```



شكل (١١-١) : فى هذا الباب dialog box يستخدم components الملف والدليل الاربعة  
ل Delphi DirectoryListBox ، DriveComboBox ، FileListBox ، و FilterComboBox



يوضح Delphi قوائم الملف والدليل الحالية فى وقت التصميم ، ولكن يجب عليك تشغيل البرنامج لاختيار ملفات وادلة جديدة ، ولتحريك النوافذ .

لإظهار المسار الذى تم اختياره حالياً ك string ، أضف Label object على ال form (عادة ، يجب ان يكون فوق ال DirectoryListBox) . قم بتعيين اسم ال Label الخاصية ال DirLabel التابعة لل DirectoryListBox . أو ، لتقوم بالربط اثناء وقت التشغيل ، أدخل هذه العبارة فى ال OnCreate الخاص بال form :

DirectoryListBox1.DirLabel := Label1;

يقوم ال DirectoryListBox بتعيين مسارات الدليل ، بما فى ذلك ال drive الحالي ، ليضاف الى ال Caption الخاص بال Label . لعرض مسارات متداخلة يع استبدل ال component الادلة الجذرية الاساسية بزر بيضاوى . هذا يجعل ال Label قصير نسبياً (حوالى ٢٤ حرف كحد أقصى ، أو عرض ال DirectoryListBox تقريباً) . على سبيل المثال ، يعرض ال DirectoryListBox هذا المسار :

C:\Program Files\Borland\Delphi 4\Demos\COOLSTUFF

على انه ال Caption المختصر :

C:\...\Demos\COOLSTUFF

خاصية ال Drive تعطى حرف ال drive المختار حالياً . على سبيل المثال ، يحدد حرف ال C على انه حرف ال drive :

C := DirectoryListBox1.Drive;

استخدم خاصية ال Directory لتحديد المسار الذى يتم اختياره حالياً . لعرض اسماء مسارات كاملة غير مختصرة بزر بيضاوى ، لا تحدد اسم Label الخاصية ال DirLabel . بدلاً من ذلك ، أضف Label على ال form وفى OnChange الخاص بال DirectoryListBox ، قم بتعيين ال Directory string الخاص بال Caption الخاص بال Label .

## : DriveComboBox

إذا لم تكن تريد السماح بالوصول الى drive معين فقط ، فإنك غالباً تريد إضافة الـ DriveComboBox وربطه بالـ DirectoryListBox . أضف DriveComboBox على الـ form (غالباً ما يكون أسفل الـ form) . عين إسم DriveComboBox الخاصية الـ DirList التابعة للـ DriveComboBox ، أو استخدم هذه العبارة فى الـ OnCreate الخاص بالـ form :

```
DriveComboBox1.DirList := DirectoryListBox1;
```

عندما يختار المستخدمون drive مختلف ، يقوم الـ DriveComboBox تلقائياً بتحديث شجرته . إذا ربطت أيضاً الـ FileListBox بالـ DirectoryListBox ، يتم تحديث قائمة الملف أيضاً .

إن القائمة الكاملة للـ drives المتاحة متوفرة من خلال خاصية الـ Items للـ DriveComboBox وهو object من نوع الـ TStrings . يمكنك تعيين هذه القائمة لأى خاصية TStrings أخرى ، أو لمتغير الـ TStringList . على سبيل المثال ، لعرض كل drive متاحة ، أضف الـ ListBox على الـ form ، واضف هذه العبارة لـ OnCreate الخاص بالـ form :

```
Listbox1.Items := DriveComboBox1.Items;
```

## : FileListBox

كما يمكن ان تتوقع ، يعرض الـ FileListBox اسماء ملفات فى الدليل الحالى . فى اغلب الاوقات يكون الـ DirectoryListBox مرتبط بالـ FileListBox . بهذه الطريقة ، تتغير القائمة تلقائياً عندما يتصفح المستخدمون الأدلة .

إنك غالباً ما تضيف أيضاً اثنين من الـ components : الـ Edit والـ FileListBox . أضف الـ Edit الآن (يوضح الفصل التالى كيفية استخدام الـ FileListBox) . عادة ما يكون الـ control فوق الـ FileListBox . قم بتعين الـ Name للـ Edit الخاصة الـ FileEdit التابعة للـ FileListBox ، أو أدخل هذه العبارة فى الـ OnCreate الخاص بالـ form :

```
FileListBox1.FileEdit := Edit1;
```

عندما تقوم بتشغيل البرنامج، يعرض الـ Edit أولاً الـ drive \*.\* ، الذي يختار كل الملفات . عندما يختار المستخدمون اسماء الملفات، يقوم الـ FileListBox بإدخالها في نافذة الـ Edit .

لإنشاء Read only Edit لا يستطيع المستخدمون تعديله، ولكن يستمر في عرض اختيار الملف الحالي، حدد خاصية الـ ReadOnly لـ Edit بـ True . لمنع المستخدمين أيضاً من الانتقال لـ control وإبراز نصه، حدد الـ Enabled بـ False .

إجعل الملف المختار في خاصية الـ FileName للـ FileListBox . يتضمن هذا الـ string مسار وحرف الـ drive الحالي . على سبيل المثال، لاختيار الملفات بالضغط مرتين، قم بإنشاء الـ OnDblClick للـ FileListBox ، وإدخل هذه العبارة :

```
ShowMessage(FileListBox1.FileName);
```

قم بتشغيل البرنامج واضغط مرتين اسماء الملفات لعرضها في نافذة ShowMessage . لاختيار ملفات أيضاً بضغط Enter ، قم بإنشاء الـ OnKeyPress الخاص بالـ FileListBox ، وإدخل هذا الـ code لإستدعاء الـ double Click procedure اذا كانت الـ Key للـ carriage-return مساوية للـ control code :

```
if Key = #13 then
```

```
FileListBox1.DblClick(Sender);
```

اذا كنت لا تحتاج إسم المسار كاملاً في الـ FileName string الخاص بالـ FileListBox ، قم بتمريره الى احدى الـ File-Management functions بـ ExtractFilePath ، ExtractFileName ، ExtractFileExt : Delphi . لتجربة هذه الـ functions أضف ثلاثة Labels على الـ form (يقوم Delphi بتسميتهم بـ Label2 ، Label3 ، و Label4) . قم بتعيين اجزاء من مسارات الملف المختار بهذا الـ code في الـ OnDblClick الخاص بالـ FileListBox :

```
with FileListBox1 do
```

```
begin
```

## الباب الحادي عشر : Navigating Directories and Files

```
Label2.Caption := ExtractFileExt(FileName);
Label3.Caption := ExtractFileName(FileName);
Label4.Caption := ExtractFilePath(FileName);
end;
```

يستطيع المستخدمون اختيار ملفات متعددة في FileListBox إذا كانت خاصية الـ MultiSelect له محددة بـ True . بهذا التحديد، يمكن للمستخدمين إبراز ملفات متعددة بضغط مفاتيح الـ Shift والـ Ctrl أثناء ضغط الفأرة. إذا اخترت هذا الخيار، فإنك تحتاج أن تكتب code وتدخل زر OK لتوفير قائمة الاختيار للتطبيق.

يقوم الـ Edit المرتبط بالـ FileListBox في خاصية الـ FileEdit بعرض أحدث اسم ملف تم اختياره عندما تحدد الـ MultiSelect بـ True . ولكن بدلاً من الـ Edit، يمكنك إضافة الـ ListBox في الـ form لعرض أسماء الملفات المختارة.

توضح القائمة (١١-١) كيفية إنشاء قائمة اختيار اسم ملف متعدد. قم بإضافة ListBox آخر في الـ form (يقوم Delphi بإعطاء اسم ListBox2)، وإدخل BitBtn بخاصية الـ BitBtn له محددة بـ bkOk. إدخل البرمجة الموجودة في القائمة (١١-١) في الـ OnClick للـ BitBtn.

**القائمة (١١-١) : الـ OnClick هذا يوضح كيفية**

**إنشاء قائمة اختيارات متعددة في الـ FileListBox**

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
  I: Integer;
begin
  ListBox2.Clear;
  if FileListBox1.SelCount > 0 then
    for I := 0 to FileListBox1.Items.Count - 1 do
      if FileListBox1.Selected[I] then
        ListBox2.Items.Add(FileListBox1.Items[I]);
    end;
```

إذا كان الـ SelCount أكبر من (0)، يكون المستخدم قد اختار اسم ملف واحد على الأقل. تستخدم الـ for loop خاصية الـ Count فى الـ Items لتحديد إجمالي أسم الملفات الذى يقوم الـ FileListBox يعرضه. لكل عنصر N، إذا كانت الـ Selected[N] بـ True، يضيف البرنامج هذا العنصر لقائمة الـ Items الخاص بالـ ListBox2. إستبدل الـ ListBox فى القائمة (١١-١) بـ TStringList object إذا كنت تحتاج قائمة اختيار اسم ملف ليست مرتبطة مثل الـ ListBox.

يمكنك تحسين فعالية الـ code الموجودة فى القائمة (١١-١) باستخدام عبارة with، ولكننى وضعت قبل الخصائص اسم الـ object حتى أوضح ما تفعله العبارات. لفعل هذا التغيير، استبدل العبارات الواقعة بعد استدعاء الـ ListBox2 لـ Clear method بما يلى:

```
with FileListBox1, Items do
  if SelCount > 0 then
    for I := 0 to Count - 1 do
      if Selected[I] then
        ListBox2.Items.Add(Items[I]);
```

### : FilterComboBox

استخدم الـ FilterComboBox لتوفير قائمة من الـ wildcard filters التى يمكن الاختيار منها مثل الـ \*.pas والـ \*.ini. قم بإضافة الـ FilterComboBox فى على الـ form (يأتى عادة تحت الـ FileListBox). لتحديث قائمة الملف الحالى عندما يختار المستخدمون الـ drive، عين اسم الـ FileListBox لخاصية الـ OnCreate للـ FilterComboBox. أو إدخل هذه العبارة فى الـ OnCreate الخاص بالـ form:

```
FilterComboBox1.FileList := FileListBox1;
```

افتح الـ Filter editor الخاص بـ Delphi بضغط الزر البيضاوى الواقع بعد خاصية الـ Filter للـ FilterComboBox. أضف الـ filters فى الـ editor بهذه الصورة:

All files (*.*)	*.*
Text files (*.Txt)	*.Txt
Pascal files (*.pas)	*.pas

## الباب الحادي عشر : Navigating Directories and Files

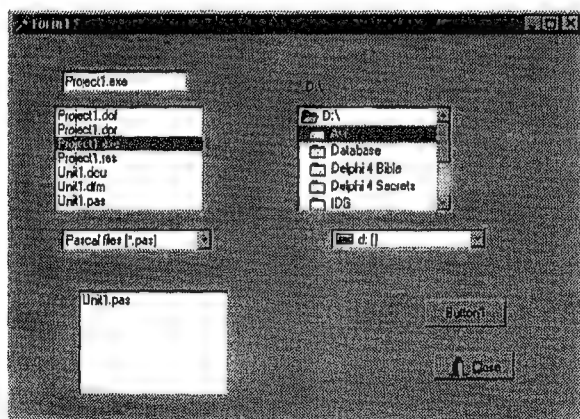
تظهر strings العمود الايسر في ال FilterComboBox. تعتبر strings العمود الايمن هي ال filters الفعلية وقد تم تمريرها لل FileListBox. اذا كان ال Edit object متعلق به، فإن اختيار filter جديد يؤدي ايضاً الى إدخاله في نافذة ال Edit.

### وضع dialogs الادلة معا:

**على القرص المدمج:** اذا كنت مستمر في المتابعة، فإن ال form تبدو مثل ال dialog المختلط في شكل (١١-٢). في هذا الفصل، تقوم بترتيب هذه الفوضى ب dialog اختيار ملف يبدو في مظهر أفضل والذي يمكنك استخدامه في كثير من العمليات من التعامل مع الدليل والملف.



توضح القائمة (١١-٢) ال source code لل DirDlg (يمكنك العثور عليها على القرص المدمج في دليل Source\DirEx). لا يوجد كثير من ال code في هذه ال module- تعتبر اغلب عمليات ال dialog تتم بصورة تلقائية وتنتج عن ربط ال objects الدليل والملف.



شكل (١١-٢)، الأولى dialog box (والفوضوي) الذي تم إنشاؤه في الفصول السابقة موضحاً components دليل وملف Delphi

القائمة (١١-٢) ال DirDlg module في الدليل الفرعي DirEx  
unit Dirdlg;

```
interface
```

```
uses
```

```
  SysUtils, Windows, Messages, Classes, Graphics,  
  Controls, Forms, Dialogs, StdCtrls, Buttons, FileCtrl;
```

```
type
```

```
  TDirDlgForm = class(TForm)  
    FileListBox: TFileListBox;  
    DirectoryListBox: TDirectoryListBox;  
    DriveComboBox: TDriveComboBox;  
    FilterComboBox: TFilterComboBox;  
    FileNameEdit: TEdit;  
    FileNameLabel: TLabel;  
    DirectoriesLabel: TLabel;  
    DirLabel: TLabel;  
    ListFilesLabel: TLabel;  
    DrivesLabel: TLabel;  
    OkBitBtn: TBitBtn;  
    CancelBitBtn: TBitBtn;  
    procedure FileListBoxDblClick(Sender: TObject);
```

```
  private
```

```
    { Private declarations }
```

```
  public
```

```
    { Public declarations }
```

```
  end;
```

```
var
```

```
  DirDlgForm: TDirDlgForm;
```

```
implementation
```

```
{ $R *.DFM }
```

## الباب الحادى عشر : Navigating Directories and Files

```
procedure TDirDlgForm.FileListBoxDbClick(Sender: TObject);
begin
    OkBitBtn.Click;
end;
```

```
procedure TDirDlgForm.DriveComboBoxChange(Sender: TObject);
begin
    DirectoryListBox.Drive := DriveComboBox.Drive;
    FileListBox.Drive := DriveComboBox.Drive;
    FileListBox.Directory := DirectoryListBox.Directory;
end;
```

end.

يوضح برنامج اختياري في دليل ال DirEx كيفية استخدام ال DirDlg dialog . تقوم ال ShowModal function الخاصة بالform بإدخال قيمة ال ModalResult ، والتي يمكن للتطبيق ان يفحصها ليتعرف ال dialog على سبيل المثال ، تعرض القائمة ( ١١ - ٣ ) ، المأخوذة من البرنامج الاختياري بال DirEx ، إسم ملف مختار في ال ShowMessage dialog اذا كانت ال ShowModal تدخل mrOk . لكي يحدث هذا ، تقوم ازرار ال OK و ال Cancel الخاصة بال dialog بتحديد خصائص ال ModalResult لها بـ mrOk و mrCancel على التوالي . يمكنك ايضاً تعيين قيم لل ModalResult في وقت التشغيل بعبارة مثل :

```
ModalResult := mrOk;
```

القائمة (١١-٣) : ال ShowMessage يعرض

file - Selection dialog [راجع شكل (١١-١)]

```
procedure TMainForm.TestBitBtnClick(Sender: TObject);
begin
    with DirDlgForm do
        if ShowModal = mrOk then
            ShowMessage('Selected file = ' + FileNameEdit.Text);
end;
```



## تطوير الـ Directory - Based Utilities

عندما تقوم بإنشاء forms مثل Selection dialog - file في الفصل السابق، فإنك قد تحتاج أن تستخدمها في تطبيقات أخرى. لتنظيم الـ source code لتطبيقك، من الأفضل دائماً أن تنشئ دليلاً منفصلاً لكل module لكل برنامج. يمكن أن يستخدم برنامجاً واحداً أو أكثر هذه الـ modules بإضافتها إلى المشروع. كمثال على هذا المفهوم، ولتوضيح كيفية استخدام الـ dialog file Selection في الـ utility، يقوم الـ DirExec، بمضاعفة إمكانيات الـ Explorer ليقوم بتشغيل أى ملف الـ executable code. يستطيع البرنامج أيضاً فتح أى ملف في قاعدة بيانات سجل الـ Windows. على سبيل المثال، يفتح الـ DirExec الـ Notepad لعرض text file والذي ينتهى بالإمتداد .txt.

**على القرص المدمج:** حاول إنشاء هذا التطبيق بنفسك - وهذا يمنحك الخبرة في استخدام الـ Project Manager الخاص بـ Delphi لإقامة مشروعات متعددة لتطوير الـ module. قم بإنشاء دليل لبرنامجك، وابدأ مشروعاً جديداً. انظر الـ Project Manager، واضغط زر الـ Add. إنتقل إلى نسخة من دليل الـ Source\DirEx من على القرص المدمج، واختر ملف الـ DirDlg.pas file. هذا يضيف الـ DirDlg module لمشروع البرنامج. يجب أيضاً أن تضيف اسم الـ module unit لأمر الـ uses الخاص بالبرنامج المضيف. يمكنك عندئذ إضافة أزرار أو أوامر المشروع لفتح الـ dialog لاختيار أسماء ملفات. استخدم القائمة (١١-٤) كمرشد لك. بعد القائمة، سوف أوضح كيف يقوم البرنامج بتشغيل التطبيقات وفتح الملفات المسجلة. يوضح الشكل (١١-٣) عرض البرنامج.

### القائمة (١١-٤): DirExec\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
SysUtils, Windows, Messages, Classes, Graphics,
```

## Navigating Directories and Files : الباب الحادى عشر

Controls, Forms, Dialogs, StdCtrls, Buttons, ShellAPI,  
DirDlg;

type

```
TMainForm = class(TForm)
  RunBitBtn: TBitBtn;
  BitBtn2: TBitBtn;
  procedure RunBitBtnClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

var

```
MainForm: TMainForm;
```

implementation

```
{ $R *.DFM }
```

```
function ExecuteFile(const FileName, Params, Dir: String;
  ShowCmd: Integer): THandle;
```

```
begin
```

```
  Result := ShellExecute(Application.MainForm.Handle,
  nil,
  PChar(FileName), PChar(Params), PChar(Dir),
  ShowCmd);
```

```
end;
```

```
procedure TMainForm.RunBitBtnClick(Sender: TObject);
```

```
begin
```

```
  with DirDlgForm do
```

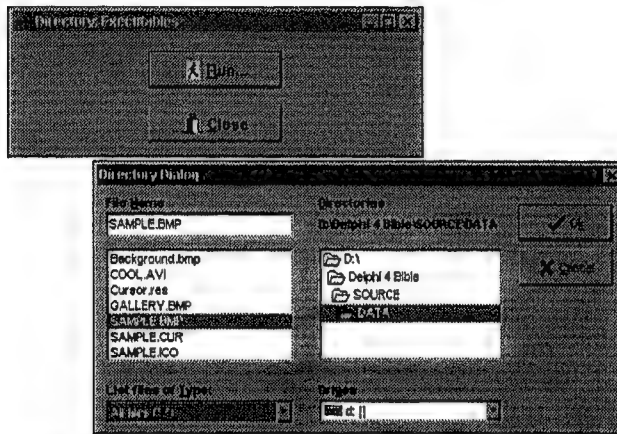
```
    if ShowModal = mrOk then
```

```

if ExecuteFile(FileNameEdit.Text, "
    DirectoryListBox.Directory,SW_SHOW) <= 32 then
    MessageDlg('Unable to open file or program',
        mtError, [mbOk], 0)
    else
    Application.Minimize;
end;

end.

```



شكل (١١-٣): يوضح تطبيق الـ DirExec كيفية استخدام dialog اختيار الملف الخاص بهذا الباب لتشغيل برامج أخرى وفتح ملفات مسجلة

فى الباب السابق، عرفت كيفية استخدام الـ ExecuteFile function الخاصة بالـ unit FMXUtils لتشغيل برامج أخرى. (توجد الـ unit فى دليل الـ Demos\Doc\Filemanex الخاص بـ Delphi). تقوم هذه الـ function باستدعاء الـ Windows API function، وهى الـ ShellExecute، وهى متاحة دون استخدام الـ FMXUtils function. يمكن للـ ShellExecute أيضاً أن تفتح أى ملف مسجل - على سبيل المثال، تلك الملفات التى لها أسماء تنتهى بـ .txt و .bmp.

مثال الـ DirExec [راجع القائمة (١١-٤)] يتضمن الـ function، وهى الـ ExecuteFile، والتى يمكنك قصها ولصقها فى أى تطبيق. تقوم الـ function باستدعاء الـ ShellExecute بخمسة parameters:

## الباب الحادي عشر: Navigating Directories and Files

\* الـ handle التابع للتطبيق الحالي (موجود في خاصية الـ Handle في الـ MainForm الخاص بالـ Application).

\* للعملية الاختيارية (حدد الـ 'Print' هنا لطباعة ملف بدلاً من فتحه).

\* ثلاث null-terminated strings.

لأن الـ ShellExecute يعتبر Windows API function ، فلا يمكنك تغذيته بـ Pascal strings. تمثل الـ string الثلاثة: اسم الملف، أي parameters، ومسار الدليل.

يقوم الـ ShellExecute بإدخال الـ handle الحالي للتطبيق المفتوح. إذا كان صالحاً، فإن هذه القيمة تكون أكبر من ٣٢، وهذه الحقيقة يستخدمها الـ OnClick لزر الـ Run الخاص بالبرنامج الاختياري لعرض رسالة خطأ. لاحظ الاستدعاء للـ Application.Minimize قرب نهاية القائمة. هذه الخطوة اختيارية، ولكن تزيح نافذة البرنامج بعيداً عن طريق التطبيق المختار.

### ملفات Drag-and-Drop:

أي برنامج يستطيع عرض ملف نص يجب أن يتعامل مع خدمات الـ Drag-and-Drop حتى يتمكن المستخدمون من سحب أسماء الملفات من الـ Windows File Manager وإسقاطها في الـ Memo أو أي component آخر بالتطبيق. إن Delphi لا يقدم خدمات الـ drag-and-drop مباشرة، ولكن يمكنك إضافة هذه الميزة ببذل القليل من الجهد.

الخطوة الأولى هي أن تخبر الـ Windows أن نافذة برنامجك يمكن أن تقبل ملفات الـ drag-and-drop افعل هذا في الـ OnCreate الخاص بالـ form بإضافة هذه العبارة:

DragAcceptFiles(Handle, True);

وايضاً، أضف الـ ShellAPI unit في أمر الـ uses للـ module. يقوم الـ ShellAPI بتعريف الـ DragAcceptFiles والـ parameter الخاصة بها:

procedure DragAcceptFiles(Wnd: HWND; Accept: Bool);

• **Wnd**: handle للنافذة الحالية، وهو الذى يحدد النافذة التى تتقبل عملية الإسقاط والسحب للملفات غالباً، يجب ان تمرر خاصية الـ Handle الخاصة بال form التى تتقبل إسقاط الملف.

• **Accept**: حدد هذا ال parameter بـ True لإخبار ال Windows ان هذه النافذة تقبل اسماء الملف التى يتم سحبها واسقاطها. حدد هذا ال parameter بـ False لفصل عمليات السحب والاسقاط. يوصى الـ Microsoft بالقيام بهذا قبل إنتهاء البرنامج - فى تطبيقات Delphi، يمكنك إيقاف السحب والاسقاط فى ال OnCreate الخاص بال form.

فى أية نافذة تستدعى ال DragAcceptFiles، عندما تقوم بسحب اسم ملف من ال Windows File Manager الى ذلك النافذة وتطلق زر الفأرة، يرسل ال Windows الى النافذة رسالة ال wm\_DropFiles. ومع الرسالة يأتى اسم الملف لتلقى هذه الرسالة، ادخل ال event handler فى القطاع ال private لل form class الخاصة بك. (يمكن ان يذهب التعريف الى أى مكان بال class، ولكنه غالباً ما يكون فى القطاع ال private أو ال protected. على سبيل المثال، اضع السطور التالية بين القطاعين ال private وال public فى تعريف ال form class:

```
protected
procedure WMDropFiles(var Msg: TMessage);
message wm_DropFiles;
```

تكون ال Protected procedures من نوع ال never-never land بين القطاعين ال private and public. يمكن للـ class method وال method فى ال classes المشتقة فقط ان تصل إلى ال protected methods. وهذا يختلف عن التعريف ال private، والتى يمكن ان يصل إليها اعضاء ال class فقط، ولكن ليس لأى classes مشتقة. تستطيع ايه عبارة الوصول لقطاع ال Public لل class كالغسيل المنشور بين البيوت. العناصر ال protected فهى فى الردهة- يجب ان يكون داخل البيت حتى تحصل عليها. اما التعريف ال private فهى فى حجرتك الخاصة. فطالما ابقيت الباب مغلقاً بقل، فلا يستطيع احد ان يسهم الا انت.

## الباب الحادى عشر : Navigating Directories and Files

قم بتنفيذ ال WMDropFiles method لأداء اعمال عندما يقوم المستخدمون باسقاط اسم ملف على النافذة . يجب ان يكون لل procedure ال format العامة الموضحة فى القائمة (١١-٥) . يشير التعليق الى المكان الذى يمكنك ان تؤدي فيه اعمالاً مع اسماء الملفات التى يتم اسقاطها .

```
القائمة (١١-٥): نفذ ال WMDropFiles method بهذا النوع من ال code
{ Handle wm_DropFiles message }
procedure TMainForm.WMDropFiles(var Msg: TMessage);
var
  Filename: array[0 .. 256] of Char;
begin
  DragQueryFile(
    THandle(Msg.WParam),
    0,
    Filename,
    SizeOf(Filename));
    { ... Perform action with dropped filename }
  DragFinish(THandle(Msg.WParam));
end;
```

فى ملفات ال WMDropFiles ، يحمل null-terminated string ، وهو ال FileName ، اسم الملف المسقط . يمكنك ان تجعل هذا ال string اطول اذا لزم الأمر ، ولكن ال ٢٥٦ رمزاً يجب ان يكونوا كافيين حتى لاكثر اسماء المسارات المتداخلة عمقاً . استدع ال DragQueryFile للحصول على المعلومة المسقطة . يعرف ال ShellAPI هذه ال function بـ :

```
function DragQueryFile(Drop: THandle; FileIndex: Word;
  FileName: PChar; cb: Word): Word;
```

● **Drop handle** : ال structure الداخلى الذى يحتوى معلومات عن اسماء الملفات التى تم سحبها وإسقاطها . غالباً يجب ان تمرر الى ال Drop حقل ال THandle من رسالة ال WMDropFiles الملفاه فى ال object .

● **FileIndex**، حدد هذا ال parameter بـ \$FFFF (يساوى 1- بالكسر العشرى) لطلب عدد اسماء الملفات المسقطة . قم بتمرير قيمة من صفر الى عدد اسماء الملفات المسقطة ناقص واحد لنسخ الاسماء فى ال FileName parameter .

● **Filename** : Char array كبير بما يكفى لان يحمل اسم مسار كامل ، عادة ٢٥٦ بايت على الأقل . إدخال null-terminated string فى ال array ، أو قم بتمرير nil لطلب عدد الملفات المسقطة . قم بتمرير nil ايضاً لطلب حجم ملف بعينه بال bytes .

● **cb**، حجم ال Filename Buffer بال bytes .

إذا قمت بتمرير (-1) لـ FileIndex parameter ، تقوم ال DragQueryFile بإدخال عدد الملفات المسقطة . قم بتعريف متغير Word لحمل نتيجة ال function :

var

NumFiles: Word;

بعد ذلك ، استدع ال DragQueryFile لتحديد عدد اسماء الملفات التى تم إسقاطها فى النافذة :

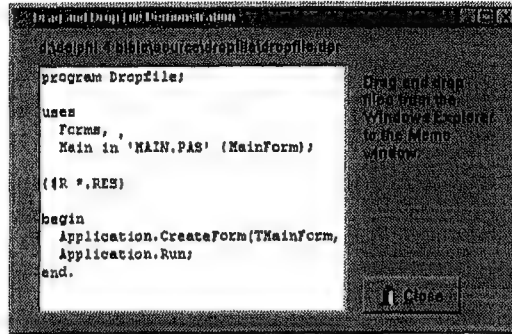
```
NumFiles := DragQueryFile(
    THandle(Msg.WParam),
    $FFFF,
    nil,
    0);
```

● **على القرص المدمج**: فى كثير من الحالات ، انك تحتاج ان تقبل اسم ملف واحد فقط . كمثال عملى على هذا ، يوضح تطبيق ال DropFile على القرص المدمج (فى دليل ال Source\DropFile) كيفية برمجة Memo ليقبل اسماء ملفات السحب والاسقاط . قم بتشغيل ال DropFile واسحب أى اسم ملف نص من ال Windows File Manager الى نافذة ال Memo بالبرنامج ، والتى تقرأ الملف من القرص وتعرض سطره .



## الباب الحادي عشر : Navigating Directories and Files

يوضح شكل (٤-١١) عرض البرنامج . توضح القائمة (١١-٦) . الـ source code.



شكل (٤-١١) : برنامج الـ DropFile يوضح ملف مشروع الـ DropFile.dpr، والذي قمت بسحبه لناقذة الـ Memo من الـ Windows File Manager

القائمة (١١-٦) : DropFile\Main.pas

unit Main;

interface

uses

SysUtils, Windows, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, StdCtrls, Buttons, ShellAPI;

type

TMainForm = class(TForm)

    Memo1: TMemo;

    BitBtn1: TBitBtn;

    Label1: TLabel;

    FileNameLabel: TLabel;

    procedure FormCreate(Sender: TObject);

    procedure FormDestroy(Sender: TObject);

private

{ Private declarations }

protected



```

procedure WMDropFiles(var Msg: TMessage);
    message wm_DropFiles;
    public
    { Public declarations }
    end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

{ Handle wm_DropFiles message }
procedure TMainForm.WMDropFiles(var Msg: TMessage);
var
    Filename: array[0 .. 256] of Char;
begin
    DragQueryFile(
        THandle(Msg.WParam),
        0,
        Filename,
        SizeOf(Filename));
    with FileNameLabel do
    begin
        Caption := LowerCase(FileName);
        Memo1.Lines.LoadFromFile(Caption);
    end;
    DragFinish(THandle(Msg.WParam));
end;

{ Tell Windows this window can accept drag-and-drop files }
procedure TMainForm.FormCreate(Sender: TObject);

```

## الباب الحادى عشر : Navigating Directories and Files

```
begin
  DragAcceptFiles(Handle, True);
end;

{ Disable drag-and-drop files (recommended) }
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  DragAcceptFiles(Handle, False);
end;

end.
```

لتشغيل سحب واسقاط اسماء الملف، يقوم OnCreate الخاص الـ DropFile form باستدعاء الـ DragAcceptFiles. لإلغاء السحب والاسقاط، يقوم الـ OnDestroy procedure باستدعاء نفس الـ function، ولكنه يمرر False للـ procedure الثانى.

تعرف الـ TMainForm class الـ handler الخاص برسالة الـ WMDropFiles. عندما يرسل الـ Windows رسالة الـ WMDropFiles للنافذة، تحتوى الـ parameter Msg على المعلومة التى تم تمريرها. يستخدم الـ procedure implementation هذه المعلومة فى استدعاء الـ WMDropFile للحصول على اسم الملف المسقط. ولأن هذا الاسم null-terminated string يقوم البرنامج بتمريره للـ StrPas لتحويل النص الى Pascal string، ويتم تعيينه للـ Caption الخاص بالـ FileNameLabel وتمريره لـ LoadFromFile method التابع لـ Memo1. إننى أضيف استدعاءً للـ LowerCase لتحويل اسم الملف الى الحروف الصغيرة، والتى هى أسهل على العين.

وهناك مشكلة واحدة مع هذه التقنيات وهى أنها لا تعمل عند سحب اسماء ملف على أيقونة التطبيق وهو فى حالة الـ minimized والسبب هو أن نافذة الـ form الرئيسية لـ Delphi تكون غير مرئية عند تصغير البرنامج. لتنفيذ السحب والإسقاط لهذه الحالة، يجب أن تضيف code للتطبيق. هذه هى النافذة التى توضح الأيقونة المصغرة- إنها ليست نفس النافذة المستخدمة من قبل الـ form.

اتبع هذه الخطوات لإقامة سحب وإسقاط لأيقونة مصغرة الى تطبيق ال DropFiles . أضف استدعاءً ثانياً لل DragAcceptFiles فى OnCreate الخاص بال form . تخبر العبارة Windows أن نافذة ال Application يمكن أن تقبل ملفات السحب والإسقاط :

```
DragAcceptFiles(Application.Handle, True);
```

لإبطال السحب والإسقاط لهذه النافذة، أضف عبارة مشابهة لـ OnDestroy لل form ، ولكن قم بتمرير False لـ DragAcceptFiles parameter الثانى :

```
DragAcceptFiles(Application.Handle, False);
```

لا يتعامل ال Application object مع الرسائل بنفس طريقته مع ال form class . تصل الرسائل فى حالة أولية، بالضبط كما تم تمريرها بواسطة ال Windows ، وإنك تحتاج الى ال procedure خاصاً للتعامل معها . قم بتعريف ال procedure كما يلى (اسمه لا يهم) فى القطاع protected لل form class . على سبيل المثال، أضف هذا التعريف تحت ذلك الخاص بال WMDropFiles :

```
procedure AppOnMsg(var Msg: TMsg; var Handled: Boolean);
```

يعتبر ال Parameter Msg من نوع ال TMsg ، والذي يتم تعريفه بواسطة Delphi فى ال Windows.pas :

```
tagMSG = packed record
hwnd: HWND;
message: UINT;
wParam: WPARAM;
lParam: LPARAM;
time: DWORD;
pt: TPoint;
end;
TMsg = tagMSG;
```

## الباب الحادى عشر : Navigating Directories and Files

إنه فى هذه ال form تصل الرسائل فى ال Application . فى هذه الحالة ، نحن نهتم برسالة واحدة فقط ، وهى wm\_DropFiles . بالإضافة الى ذلك ، نريد أن نميز أن هذه الرسالة فقط عند تصغير التطبيق . توضح القائمة (١١-٧) كيفية كتابة ال handler .

يقوم ال AppOnMsg procedure بفحص ال Message.Msg لرسالة ال wm\_DropFiles . اذ كان التطبيق قد تم تصغيره أيضاً أيضاً (يقوم ال IsIconic بإرجاع True) ، يستدعى ال procedure DragQueryFile للحصول على اسم ملف وتحميل نصه فى ال Memo object للبرنامج .

القائمة (١١-٧): أدخل برنامج ال handler هذا فى DropFiles لتشغيل ملفات السحب والإسقاط للأيقونة المصغرة للتطبيق

```
procedure TMainForm.AppOnMsg(var Msg: TMsg; var
Handled: Boolean);
var
  Filename: array[0 .. 256] of Char;
begin
  with Application do
    if (Msg.Message = wm_DropFiles) and IsIconic(Handle)
    then
      begin
        DragQueryFile(THandle(Msg.WParam),
          0, Filename, SizeOf(Filename));
        with FileNameLabel do
          begin
            Caption := LowerCase(FileName);
            Memo1.Lines.LoadFromFile(Caption);
          end;
        DragFinish(THandle(Msg.WParam));
      end;
    end;
end;
```

هناك خطوة إختيارية تجعل هذا ال code يعمل بشكل جيد . يجب أن تعين ال handler (وهو ال AppOnMsg فى هذا المثال) لل OnMessage الخاص بال

Application. لضمان أن البرنامج يتلقى كل الرسائل المستهدفة، افعل هذه الخطوة مبكراً- في الـ OnCreate للـ form، مثلاً. قم بإنشاء برنامج للـ OnCreate الخاص بالـ DropFile، وأدخل هذه العبارة:

Application.OnMessage := AppOnMsg;

يمكنك الآن تشغيل الـ DropFile وتصغير نافذته على الحاسب المكتبي للـ Windows. اسحب اسم ملف نص من الـ File Manager واسقطه على الأيقونة. عندما تفتح نافذة الـ DropFile، سوف ترى نص الملف في الـ Memo object.

### قراءة وكتابة ملفات الـ .ini :

تبسط الـ TIniFile class الخاصة بـ Delphi العمل الشاق الخاص بقراءة وكتابة ملفات بدء الـ Windows المعرفة بالامتداد الـ .ini. لا الـ TIniFile class لا تعتبر component، وهي لا تظهر في لوحة الـ VCL لـ Delphi. لهذا السبب، يجب عليك إنشاء وتحرير الـ TIniFile objects، واستدعاء methods في ظل تحكم البرنامج كاملاً.

في غالبية البرامج، تحتاج الى two procedures لقراءة وكتابة مواصفات ملف الـ .ini. على سبيل المثال، يمكنك حفظ خيارات ومواصفات أخرى، التي يستطيع المستخدمون الخبراء تعديلها بتحميل ملف الـ .ini في text editor. لإنشاء وتحديث الملف، أنشئ أولاً TIniFile objects. قم بتعريفه كما يلي:

var

IniFile: TIniFile;

ثم، في قلب الـ procedure، أنشئ IniFile object بإستدعاء الـ Create method الخاص بالـ TIniFile:

IniFile := TIniFile.Create('YourProg.ini');

عندما تنتهي من استخدام الـ TIniFile، حدده بهذه العبارة:

IniFile.Free;

إذا كنت لم تذكر معلومات عن اسم المسار والـ drive لتغيير اسم الملف، يتم إنشاء ملف الـ .ini في دليل الـ Windows. هذا هو الأفضل دائماً، ولكن بعض

## الباب الحادى عشر : Navigating Directories and Files

المبرمجين ينشئون ملفات الـ ini. فى دليل الخاص بالتطبيق . اينما تضع الملف ، بعد إنشاء الـ IniFile object ، استدع الـ method لقراءة وكتابة مواصفات البدء . يجب ان يكون كل صفه فى القطاع المحاط بالاقواس ، متبوعاً بقيمة واحدة أو أكثر . على سبيل المثال ، ها هي الاسطر الثلاثة الأولى من ملف الـ SysColor.ini ، الذى تم إنشاؤه بتطبيق الـ SysColor الموضح بعد ذلك فى هذا الفصل :

```
[SysColor]
Scroll Bar=12632256
Background=4210688
```

يمكنك قراءة مواصفات ملف الـ ini. فى الـ TStringList object أو الـ TStrings بطريقتين . عرف الـ object كما يلى :

```
var
```

```
StringList: TStringList;
```

إنشئ قائمة الـ string واستدع الـ ReadSectionValues لـ IniFile object لتحميل القائمة بالإضافة إلى تنفيذ مواصفات الملف . على سبيل المثال ، استخدم عبارات مثل هذه :

```
StringList := TStringList.Create;
```

```
IniFile.ReadSectionValues('SysColor', StringList);
```

تقوم العبارة الأولى بإنشاء قائمة الـ string وتستدعى الثانية الـ ReadSectionValues التى تمرر اسم القطاع [SysColor] بلا اقواس كالـ string أو الـ TStringList object . يتضمن كل عنصر فى قائمة الـ string إسم العنصر ، علامة يساوى ، قيمة ، بلا مسافات حول علامة يساوى . بالتبادل ، لتحميل اسماء العناصر فقط ، استدع الـ ReadSection هكذا :

```
IniFile.ReadSection('SysColor', StringList);
```

يوفر الـ IniFile أيضاً methods لقراءة وكتابة قيم منفردة . على سبيل المثال ، يقوم الـ SysColor بكتابة مواصفات الـ CheckBox بالعبارة :

```
IniFile.WriteBool('Options', 'Save settings',
```

```
SaveCheckBox.Checked);
```

ال argument الاولى ، 'Options' ، تحدد القطاع الذى يذهب إليه  
العنصر . والقيمة المتغيرة الثانية هى اسم هذا العنصر ، والاخيرة هى قيمته . اذا  
كانت ال Checked محددة بـ True ، سوف تبدو سطور ملف ال ini. الناتج هكذا :  
[Options]

Save settings=1

لقراءة المواصفات ، استدع احدى Read methods الخاصة بال IniFile  
class . على سبيل المثال ، يقوم ال SysColor بتحميل قيمة ال CheckBox  
لتحديد ال Save بالعبرة :

SaveCheckBox.Checked :=

IniFile.ReadBool('Options', 'Save settings', False);

تعمل methods القراءة والكتابة الأخرى - مثل ال ReadInteger ،  
ReadString ، WriteInteger ، و WriteString - بطريقة مشابهة ، ولكن إقرأ  
واكتب قيم للأنواع المشار إليها .

بعد قراءة مواصفات ملف ال ini. فى قائمة ال string يمكنك وضع  
مواصفات معينة . افعل هذا بفهرسة ال Values array لقائمة ال string المستهدفة .  
على سبيل المثال ، اذا كانت ال T هى TStringList object تم تحميله بواسطة ال  
ReadSectionValues method لـ IniFile object ، إذن ال Values[S]  
تساوى قيمة ال string للعنصر المعروف بـ S . يمكنك تعيين هذه القيمة لـ string :

S := T.Values['Button color'];

ولكن ، إنك تريد غالباً أن تحولها قيمة عدد صحيح أو أى قيمة ثنائية أخرى  
للاستخدام فى البرنامج . افعل هذا باستدعاء function تقوم بمثل هذا مثل ال  
: StrToInt

K := StrToInt(T.Values['Button color']);

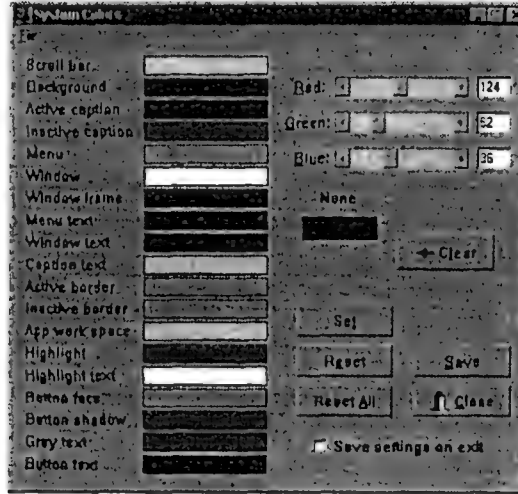
## : SysColor Utility

على القرص المدمج: ان البرنامج والطول الكامل ، SysColor ، يضع  
هذه المفاهيم موضع التنفيذ . ان ال source code طويلة الى حد ما



## الباب الحادي عشر: Navigating Directories and Files

بحيث يصعب ذكرها هنا كاملة ، لذا ساقدم اجزاء منها فى حينه . ويتبع التوضيح ، بالطبع ، كل جزئية قائمة . توجد ال source code الكاملة ، بالطبع ، على القرص المدمج فى دليل ال Source\SysColor . يوضح شكل (١١-٥) البرنامج راجع الفقرة " كيفية استخدام ال SysColor " لمعرفة التعليمات .



شكل (١١-٥)، يعرض ال SysColor ويمكنك من اختيار ١٩ لون لنظام ال Windows ، مثل لون خلفية النوافذ، ولون عنوان النافذة ال active وال inactive

### كيفية استخدام ال SysColor

بما انك درست ال SysColor، فانت مستعد لاستخدامه لاحظ ان المستطيلات الملونة تعرض مواصفات اللون الحالى. اضغط أى مربع لون لاختياره وتنقل لونه الى مربع العينة الكبير، الذى يحمل عنوان None. يظهر المربع اللون المختار واسمه. اضغط Click لفك ارتباط مربع العينة بأى لون مختار (هذا جيد فى اختيار قيم الالوان دون التأثير على صفة ال labeled).

استخدم ال Red scroll bars، Green، و Blue لتعديل قيم الالوان لـ Windows. على سبيل المثال، غير تعليق ال Active للون الاحمر النادى واضغط Set. انك ترى تعليق النافذة ال active يتغير لخلفية حمراء.

اضغط زر Reset لإعادة الالوان المحددة فى المواصفات الاصلية عندما تبدأ البرنامج. اتبع هذا بضغط Set اذا كنت تريد إعادة الالوان نظام ال Windows الى قيمتها الاصلية.

"تابع"



اضغط زر الـ Reset All لإعادة تحديد الالوان المحددة بقيم البدء الخاصة بها وكذلك استعادة الوان نظام الـ Windows. يؤدي هذا الزر نفس الأعمال التي تتم عند ضغط الـ Reset متبوعاً بالـ Set.

اضغط زر الـ Save لحفظ اختيارات الالون الحالية في الـ SysColor.ini، الموجود في دليل الـ Windows. في المرة التالية التي تقوم فيها بتشغيل الـ SysColor، فهو يعرض القيم المحفوظة في مستطيلات محددة. إذا لم تكن تريد استخدام الوان محفوظة، احذف الـ SysColor.ini وادخل البرنامج، الذي يعرض قيم لون الـ Windows الحالية.

اضغط زر الـ Close لإنهاء البرنامج (أو يمكنك اختيار أمر الـ FileExit). إذا كانت مواصفات الـ Save على الـ check box الخاص باختيار الخروج قد تم تشغيلها، يقوم البرنامج بتحديث الـ SysColor.ini عندما تتركه. اجعل الـ check box الخاص بالاختيار الـ disable إذا لم تكن تريد حفظ اختيارات لونك تلقائياً.

توضح القائمة (١١-٨) قطاع الـ interface للـ SysColor. ان الـ Label والـ Shape المرقمة تعرض مستطيلات الالون المحددة في النافذة. (يقدم الباب الثالث عشر الـ Shape وتقنيات جرافيك أخرى .

القائمة (١١-٨)، (قطاع interface) SysColorMain.pas

unit Main;

interface

uses

SysUtils, Windows, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, StdCtrls, Buttons, ExtCtrls,  
AboutDlg, IniFiles, Menus;

const

maxColors = 19;      { Number of Windows system colors }  
  redMask = \$000000FF;      { Red value extraction mask }  
}

## Navigating Directories and Files: الباب الحادي عشر

```
greenMask = $0000FF00;    { Green value extraction mask }
blueMask = $00FF0000;    { Blue value extraction mask
}
iniFileName = 'SysColor.ini'; { In the Windows directory
}
```

type

```
TMainForm = class(TForm)
    BlueEdit: TEdit;
    BlueLabel: TLabel;
    BlueSB: TScrollBar;
    ClearBitBtn: TBitBtn;
    CloseBitBtn: TBitBtn;
    ColorEdit: TEdit;
    ColorLabel: TLabel;
    FileAbout: TMenuItem;
    FileExit: TMenuItem;
    FileMenu: TMenuItem;
    GreenEdit: TEdit;
    GreenLabel: TLabel;
    GreenSB: TScrollBar;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
```

```

Label14: TLabel;
Label15: TLabel;
Label16: TLabel;
Label17: TLabel;
Label18: TLabel;
Label19: TLabel;
MainMenu1: TMainMenu;
N1: TMenuItem;
RedEdit: TEdit;
RedLabel: TLabel;
RedSB: TScrollBar;
ResetAllBitBtn: TBitBtn;
ResetBitBtn: TBitBtn;
SaveBitBtn: TBitBtn;
SaveCheckBox: TCheckBox;
SetBitBtn: TBitBtn;
Shape1: TShape;
Shape2: TShape;
Shape3: TShape;
Shape4: TShape;
Shape5: TShape;
Shape6: TShape;
Shape7: TShape;
Shape8: TShape;

```

(continued)

Listing 11-8: (continued)

```

Shape9: TShape;
Shape10: TShape;
Shape11: TShape;
Shape12: TShape;
Shape13: TShape;
Shape14: TShape;
Shape15: TShape;

```

```
Shape16: TShape;
Shape17: TShape;
Shape18: TShape;
Shape19: TShape;
procedure ClearBitBtnClick(Sender: TObject);
procedure EditChange(Sender: TObject);
procedure FileAboutClick(Sender: TObject);
procedure FileExitClick(Sender: TObject);
procedure FormClose(Sender: TObject);
var Action: TCloseAction;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure ResetAllBitBtnClick(Sender: TObject);
    procedure ResetBitBtnClick(Sender: TObject);
    procedure SaveBitBtnClick(Sender: TObject);
    procedure SaveCheckBoxClick(Sender: TObject);
    procedure SBChange(Sender: TObject);
    procedure SetBitBtnClick(Sender: TObject);
    procedure ShapeMouseDown(Sender: TObject);
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
private
    IniFile: TIniFile;
    IniItemList: TStringList;
    EditControls: array[0 .. 2] of TEdit;
    ScrollBars: array[0 .. 2] of TScrollBar;
    Shapes: array[0 .. maxColors - 1] of TShape;
    CurrentShape: TShape;
    procedure UpdateColor;
    procedure InitSysColorArray;
    procedure ChangeSystemColors;
    procedure SetScrollBars(C: TColor);
    procedure LoadSettings;
    procedure SaveSettings;
```

```
public
{ Public declarations }
end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}
```

يعرف البرنامج بعض الثوابت قرب بداية القائمة :

\* الـ `maxColors` تساوى عدد ألوان النظام بالـ `Windows`.

\* والـ `redMask`، `greenMask`، و `blueMask` تعتبر اعداد صحيحة عشرية ذات ٣٢ بت تستخدم لاستخلاص قيم ألوان منفردة من الـ `TColor` objects.

على سبيل المثال، التعبير `C` و `greenMask` يعزل قيمة اللون الأخضر للـ `TColor`. يعتبر الـ `iniFileName` هو اسم ملف بدء البرنامج. ولأن الاسم لا يحتوى على حرف `drive` أو أى معلومة عن مساره، يكون موضعه هو دليل الـ `Windows`.

الـ `TMainForm class` تعرف `objects` متعددة والتي تنشئ واجهة تطبيق النافذة. يتمشى رقم كل `Label` مع الـ `Shape` المقابل (على سبيل المثال، الـ `Label7` يذهب مع الـ `Shape7`). وهذه العلاقة ليست عشوائية، وكما سأوضح بعد قليل، عندما يكون لديك العديد من الـ `components` المتعلقة ببعضها، يمكنك الاستفادة من اعراف تسمية الـ `objects` بصورة تلقائية الخاصة بـ `Delphi` لكتابة `code` لـ `objects` متعلقة ببعضها. تعتبر غالبية الـ `objects` والـ `event handlers` الأخرى واضحة، لذا سوف أشرح فقط العناصر التي أضفتها للقطاع `private` للـ `TMainForm`. لقد تم إضافة ستة متغيرات وهى :

● الـ `IniFile`: هذا الـ `object` تم إنشاؤه فى `OnCreate` الخاص بالـ `form` وتم تدميره فى الـ `OnDestroy`. توفر الـ `IniFile` وصولاً إلى ملف الـ `Syscolor.ini`

## الباب الحادى عشر : Navigating Directories and Files

ل procedures متنوعة . يفتح ال object هذا الملف ويغلقه على حسب الحاجة لقراءة وكتابة مواصفاته ، لذا فمن الأكثر فعالية ان تعرف ال object فى ال form class عن ان يعرف ال TIniFile objects فى procedures منفردة . وكذلك ، اذا تم تعريفه محلياً ل procedures ، فإن ملف ال ini. سيتم فتحه واغلاقه كل مرة يتم فيها استدعاء ال procedures .

● **ال IniItemList** : هذه هى قائمة لتعريف ملف ال SysColor.ini ، والتي تخزن مواصفات اللون على انها اعداد عشرية صحيحة ذات ٣٢ بت . على سبيل المثال ، ان مدخل ال IniItemList الخاص بال SysColor.ini يحدد Background-4210688 هو ال 'Background' string . لقد وجدت انه من المساعدة ان نبقى العناصر المحددة فى قائمة string منفصلة ، والتي سوف أوضح المزيد عنها فى هذا الباب .

● **ال EditControls** : هذا ال array يسهل الوصول الى نوافذ ال Edit الثلاث الواقعة بعد ال scrollbars . (يستخدم تطبيق ال ColorDlg بالباب العاشر ايضاً هذا ال array) .

● **ال Shapes** : يحتوى هذا ال array على مراجع ل ١٩ Shape objects تعرض ألوان نظام ال Windows . ان استخدام array ييسر ال code- فمثلاً ، بدلاً من كتابة ١٩ handlers ، يمكن للبرنامج ان يستخدم ال array للإشارة الى مستطيل لون n بالتعبير Shapes[n] . كما يوضح ال SysColors ، ال Reference array object ، والتي سوف تجدها مفيدة فى كثير من الحالات .

● **ال ScrollBar** : هو array يسهل الوصول لل scrollbar object الثلاث . (تطبيق ال ColorDlg للباب العاشر يستخدم هذا ال array) .

● **ال CurrentShape** : هذا هو مرجع لمستطيل اللون الذى تم اختياره حالياً . عندما يساوى nil ، لا يتم اختيار أى شئ . بالتبادل ، كان يمكن ان استخدم فهرس عدد صحيح فى ال Shapes array للإشارة للون الحالى ، ربما 1- دون تمثيل أى اختيار . ولكن ، لان كل Delphi objects تعتبر Reference ، فمن الأفضل تعريف المتغيرات مثل ال CurrentShape التى تشير الى ال objects المستهدف

فعلياً. على سبيل المثال، يستطيع البرنامج تعيين `Shapes[n]` للـ `CurrentShape`، ثم يستخدم الـ `CurrentShape` لأداء اعمال على الـ `objects` المختار.

بالإضافة الى المتغيرات الست السابقة فإن الـ `SysColor`، يعرف ستة `procedures` خاصة فى الـ `TMainForm class`. فيما يلى تقديم مختصر لكل `procedure`- ووصف كامل عن كيفية عملهم:

• **UpdateColor**: يحدد هذا الـ `procedures` مربع اللون الكبير، الـ `Edit` object ويسمى الـ `ColorEdit`، للون الذى يمثله الـ `ScrollBars` الثلاث. يحدد الـ `procedures` ايضاً لون الـ `CurrentShape` اذا كان هناك واحداً مختاراً. وكذلك يقوم الـ `UpdateColor` بحفظ اللون الحالى فى الـ `SysColorArray` array العام.

• **InitSysColorArray**: هذا الـ `procedure` ينشئ الـ `SysColorArray` array، الذى يخزن ألوان نظام الـ `Windows` الحالية مع مجموعة اللون المعدلة حالياً. ان الـ `array` يجعل من السهل على البرنامج ان يعيد تحديد الالوان لقيم البدء الخاصة بها. يستخدم البرنامج ايضاً الـ `SysColorArray` لتحديث ملف الـ `SysColor.ini`.

• **ChangeSystemColors**: هذا الـ `procedures` يحول الى الـ `Windows` اختيارات اللون الحالى. لكل لون، تتلقى كل التطبيقات رسالة `wm_SysColorChange` التى تشير الى ان التطبيق يجب ان يقوم بتحديث نوافذه، ازراره والعناصر المؤثرة الاخرى. تقوم تطبيقات `Delphi` بهذه التحديثات بصورة تلقائية لا يجب عليك كتابة `code` لدعم رسائل الـ `wm_SysColorChange`.

• **SetScrollBars**: هذا الـ `procedures` يحدد خصائص الـ `properties` للـ `ScrollBars` الثلاث بقيم تتماشى مع لون يتم تمريره كـ `argument`.

• **LoadSettings**: هذا الـ `procedures` يقرأ ملف الـ `SysColor.ini`، ان وجد.

• **SaveSettings** : هذا ال procedures ينشئ أو يحدث مواصفات الألوان في ملف ال SysColor.ini .

**فهم procedures البدء والتعريفات ال global لل SysColor.ini :**

توضح القائمة (٩-١١) procedures البدء والتعريفات ال global لل SysColor.ini . الشرح يعقب القائمة .

القائمة (٩-١١) : SysColor\Main.pas (بدء وتعريفات)

```
type
{ Holds current system colors }
SysColorRec = record
    OriginalColor: TColor; { Color on starting program }
    CurrentColor: TColor; { New color selected by user }
end;

var
{ Array of SysColorRec values }
SysColorArray: array[0 .. maxColors - 1] of SysColorRec;

{ Update sample colors from scrollbar positions }
procedure TMainForm.UpdateColor;
begin
{ Update color edit box (the large sample window) }
    ColorEdit.Color :=
        RGB(RedSB.Position, GreenSB.Position, BlueSB.Position);
{ Update labeled Shape color box and SysColorArray }
    if CurrentShape <> nil then with CurrentShape do
        begin
            Brush.Color := ColorEdit.Color;
            SysColorArray[Tag - 1].CurrentColor := Brush.Color;
        end;
    end;
```



```

{ Load system colors into the SysColor array }
procedure TMainForm.initSysColorArray;
var
  I: Integer;
begin
  for I := 0 to maxColors - 1 do with SysColorArray[I] do
    begin
      OriginalColor := GetSysColor(I);
      CurrentColor := OriginalColor;
      Shapes[I].Brush.Color := OriginalColor;
    end;
  end;

  { Change system colors to values in SysColorArray }
  procedure TMainForm.ChangeSystemColors;
  var
    I: Integer;
    InxArray: Array[0 .. maxColors - 1] of Integer;
    ClrArray: Array[0 .. maxColors - 1] of TColor;
  begin
    for I := 0 to maxColors - 1 do
      begin
        InxArray[I] := I;
        ClrArray[I] := SysColorArray[I].CurrentColor;
      end;
    SetSysColors(maxColors, InxArray[0], ClrArray[0]);
  end;

```

يقوم نوع بيانات الـ SysColorRec بتعريف سجل من قيمتى TColor. تعتبر الـ OriginalColor هى قيمة اللون المأخوذة من الـ Windows عند البدء. تعد الـ CurrentColor هى قيمة اللون الموضحة فى مستطيل اللون المحدد. يحمل الـ SysColorArray ١٩ SysColorRec Record ، واحداً لكل لون فى نظام Windows.

## الباب الحادي عشر : Navigating Directories and Files

يقوم الـ UpdateColor Procedure باستدعاء الـ Windows RGB function لدمج خصائص الـ Position للـ ScrollBar الثلاث في الـ TColor-compatible LongInt ذي الـ ٣٢ بت . ان تعيين هذه القيمة لخاصية الـ ColorEdit object لـ ColorEdit يؤدي الى تحديث عينة اللون الكبيرة تحت الـ Scrollbars .

اذا اختار المستخدم لون محدد (الـ CurrentShape لا يساوي nil) ، تقوم الـ UpdateColor ايضاً بتعيين اللون الجديد للخاصية الفرعية الـ Brush.Color للـ CurrentShape ، ويحفظ اللون في الـ SysColorArray . لاحظ ان الـ Tag الخاص بالـ Shape يساوي رقم الـ Index التابع له في الـ SysColorArray .

يوضح الـ Procedure InitSysColorArray كيفية الحصول على تحديدات اللون الحالي من الـ Windows . تقوم الـ for loop باستدعاء الـ API function GetSysColor مع الـ I argument مساوياً لـ Index اللون . تحفظ الـ loop كل لون في الـ SysColorArray ، وتعرضه بواسطة تحديث القيمة الفرعية للـ Brush.Color للـ Shapes . هذا يعتبر نموذجاً جيداً على مدى فائدة الـ arrays المشيرة للـ objects مثل الـ Shapes .

يحول الـ ChangeSystemColors Procedur قيم اللون الـ ١٩ للـ Windows باستدعاء الـ API function SetSysColors . تتطلب هذه function ثلاثة arguments : عدد الالوان ، array للاعداد الصحيحة لـ Index اللون ، و array لقيم اللون ذات الـ ٣٢ بت . والعبرة التي تستدعي الـ SetSysColors تمرر عناوين الـ arrays والـ InxArray والـ ClrArray ، بفهرسة عناصرهم الأولى .

انظر الـ SetSysColors في مرجع الـ API online الخاصة بـ Delphi . الـ two arrays تم تعريفهما على انهما var parameters غير مكتوبة . لإعطاء عنوان array لهذا الـ parameter ، قم بتمرير عنصره الأول ، كما يوضح الـ procedure ChangeSystemColors .

### استخدام الـ sysColor's form - maintence event handlers

توضح القائمة (١١-١٠) ثلاثة من الـ handler للحفاظ على الـ form ، والتي تعد مسئولة عن بدء الـ SysColor objects ولأداء اعمال المسح عندما ينتهي البرنامج . التوضيح يعقب القائمة .

يتعامل ال FormCreate Procedure مع ال OnCreate الخاصة بال form .  
يبدء ال procedure بمتغيرات ويستدعى ال methods لإعادة ال display . توضيح ال  
for loop الأولى تقنية مفيدة فى تكوين و object Reference array ولنفحصها  
بعمق :

```
for I := 0 to maxColors -1 do
```

```
  Shapes[I] :=
```

```
    TShape(FindComponent('Shape' + IntToStr(I + 1)));
```

القائمة (١٠-١١) : SysColor\Main.pas (يوضح ال handlers)

```
{ Initialize TObject control arrays }
```

```
procedure TMainForm.FormCreate(Sender: TObject);
```

```
var
```

```
  I: Integer;
```

```
begin
```

```
{ Construct IniFile object instance }
```

```
  IniFile := TIniFile.Create(iniFileName);
```

```
{ Miscellaneous initializations }
```

```
  CurrentShape := nil; { No selected labeled color shape }
```

```
{ Assign object references to easy-access arrays }
```

```
  EditControls[0] := RedEdit;
```

```
  EditControls[1] := GreenEdit;
```

```
  EditControls[2] := BlueEdit;
```

```
  ScrollBars[0] := RedSB;
```

```
  ScrollBars[1] := GreenSB;
```

```
  ScrollBars[2] := BlueSB;
```

```
{ Assign Shape color box object references to Shapes array }
```

```
  for I := 0 to maxColors -1 do
```

```
    Shapes[I] :=
```

```
      TShape(FindComponent('Shape' + IntToStr(I + 1)));
```

```
{ Create item list from Labels for Ini file read/write }
```

```
  IniItemList := TStringList.Create;
```

```
  with IniItemList do
```

```
    for I := 0 to maxColors -1 do
```

```
      Add(TLabel(FindComponent(
```

## الباب الحادى عشر : Navigating Directories and Files

```
'Label' + IntToStr(I + 1))).Caption);
{ Load current colors and possible Ini file settings }
InitSysColorArray; { Initialize SysColor array }
LoadSettings; { Load SysColor.ini settings if present }
end;

{ Save settings if SaveCheckBox selected }
procedure TMainForm.FormClose(Sender: TObject;
var Action: TCloseAction);
begin
    if SaveCheckBox.Checked then
        SaveSettings; { Save settings in SysColor.ini }
end;

{ Last chance to clean up }
procedure TMainForm.FormDestroy(Sender: TObject);
begin
    IniItemList.Free;
    IniFile.Free;
end;
```

ان الهدف هو تعيين reference للـ Shape لكل object من الـ Shape objects، كل مستطيل يمثل لون واحد محدد فى النافذة. تقوم الـ FindComponent Function بإرجاع الـ reference لأسم object الذى تم تمريره كـ string. للإشارة الى Shape1، Shape2، . . . ، Shape19، يستخدم البرنامج الـ IntToString لتحويل متغير control زائد واحد الـ if loop الى string، يكون ملحق بالـ Shape. لاتمام التعيين، يضع البرنامج قيمة إدخال الـ FindComponent فى TLabel object. يمكنك إيجاد استخدامات متعددة للـ FindComponent خاصة عند إنشاء arrays of object reference مثل الـ Shapes.

وتقوم الـ for loop بمشابهة فى نفس الـ procedure بإنشاء قائمة للمتغيرات ملف الـ SysColor.ini. بعد إنشاء الـ IniItemList، وهو الـ TStringList object، تقوم الـ loop بإدخال الـ Labels الخاصة بالبرنامج كمواصفات ملف الـ

.ini . وهنا مرة أخرى، يقوم ال FindComponent بإرجاع reference objects للـ Label1، Label2، ...، Label19، يضع البرنامج قيمة إرجاع ال function في ال TLabel ويضع في ال IniItemList حقل ال Caption الخاص بهذا ال object. أخيراً، يقوم ال FormCreate باستدعاء ال InitSysColorArray للحصول على مواصفات اللون الحالي للـ Windows. بعد ذلك، يقوم ال LoadSettings بقراءة ال SysColor.ini، ان وجد.

يتعامل ال FormClose Procedure مع ال OnClose للـ form. يستدعى البرنامج ال SaveSettings لتحديث أو إنشاء ال SysColor.ini اذا قمت بتشغيل مواصفات ال Save عند خروج ال CheckBox. يتعامل ال FormDestroy Procedure مع ال OnDestroy. هذه هي الفرصة الأخيرة للبرنامج للقيام بالمسح قبل الانتهاء. كما هو موضح هنا، يعتبر هذا ال event فرصة جيدة لتحرير الذاكرة. المخصصة للـ objects. في هذه الحالة، يحرر البرنامج ال IniItemList object الذي انشأه ال OnCreate الخاص بالـ form.

### التعامل مع ال events للـ ScrollBar objects، ال Edit، والـ Shape،

توضح القائمة (١١-١١) كيف يتعامل ال SysColor مع ال events للـ ScrollBar، ال Edit، والـ Shape. الشرح يعقب القائمة.

القائمة (١١-١١) SysColor\Main.pas ال SysColor event handlers، ال Edit، والـ Shape

```
{ Set scrollbar positions to match color C }
procedure TMainForm.SetScrollBars(C: TColor);
begin
{ The following assignments also update Edit boxes }
  RedSB.Position := C and redMask;
  GreenSB.Position := (C and greenMask) shr 8;
  BlueSB.Position := (C and blueMask) shr 16;
end;
```

```
{ Update values in Edit boxes for ScrollBar changes }
procedure TMainForm.SBChange(Sender: TObject);
begin
```

## الباب الحادى عشر : Navigating Directories and Files

```
with Sender as TScrollBar do
    EditControls[Tag].Text := IntToStr(Position);
    UpdateColor;
end;

{ Update scrollbar positions for Edit box changes }
procedure TMainForm.EditChange(Sender: TObject);
begin
    with Sender as TEdit do
        ScrollBars[Tag].Position := StrToInt(Text);
    end;

    { Select color shape on mouse down event }
    procedure TMainForm.ShapeMouseDown(Sender: TObject;
        Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    var
        P: TLabel; { Pointer to matching TLabel object }
    begin
        CurrentShape := TShape(Sender); { Save clicked Shape }
        P := TLabel(FindComponent(
            'Label' + IntToStr(CurrentShape.Tag)));
        if P <> nil then
            ColorLabel.Caption := P.Caption; { Show color name }
            SetScrollBars(CurrentShape.Brush.Color); { Synch scroll
            bars}
        end;
    end;
```

يقوم ال SetScrollBars Procedure بتحديث خصائص ال Position لل ScrollBar الثلاثة للتطابق مع القيم الفرعية الحمراء والخضراء والزرقة فى parameter ال C الخاص بال TColor. ان تعيين قيم لل Position ينقل thumb box (ويسمى ايضاً scroll box). الى مكان قريب داخل مدى ال color. وتستخدم ثلاثة تعبيرات and منطقية ثوابت ال redMask، وال greenMask، وال blueMask لعزل القيم الفرعية فى اللون C. ان عامل ال shr (ال shift يميناً) ينقل القيم المعزولة الى المواضع الاقل اهمية فى الاعداد الصحيحة الناتجة، والتي تم تعيينها لل Position.

يوضح الباب العاشر كيف تتعامل مع الـ SBChange procedures والـ EditChange handle مع الـ OnChange events لـ ScrollBar والـ Edit. ان قيم الـ Tag الخاصة بالـ objects تساوى مواضعهم المفهرسة فى الـ arrays الـ EditControls والـ ScrollBars المشيرة للـ objects. مع هذه الـ arrays، يمكن ان تتشارك الـ Edit والـ ScrollBar فى الـ event handler ويتعاونوا، بحيث اذا قمت بإدخال قيمة فى نافذة Edit، يتتبعها الـ ScrollBar thumb box. بنفس الطريقة، ان نقل الـ thumb box يؤدي الى تحديث قيمة نافذة الـ Edit.

يتولى الـ ShapeMouseDown Procedure عملية ضغط مستطيل اللون المحدد لان الـ Shape components لا تعتبر Windows controls قابلة للاختيار، لذا ليس لها الـ OnClick events. ولكن، يمكنك تميز ضغط الفأرة لهذه الـ objects وغيرها من الـ objects الغير معروضة فى نوافذ بإنشاء الـ OnMouseDown event handler مثل الـ ShapeMouseDown.

يعين الـ Procedure اولاً للـ CurrentShape مرجعاً للـ Shape المختار، يتم تمريره الى الـ Procedure فى الـ Sender parameter. يستدعى البرنامج بعد ذلك الـ FindComponent ليضع الـ Label(n)، حيث ان الـ n تساوى قيمة الـ Tag للـ Shape. ان استخدام الـ Tags مع الـ FindComponent بهذه الطريقة تعتبر وسيلة فعالة لإنشاء علاقات فيما بين الـ objects وللإستفادة من قواعد التسمية لـ Delphi. فى المعتاد، يجب ان تسمى الـ objects طبقاً لأغراضها، ولكن فى هذه الحالة، من الأسهل ان تعمل مع الـ 38 objects Labels و Shapes باستخدام اسماءها الافتراضية، وهى Label1، Label2، ...، Shape19، Shape2، Label19، ...، Shape19.

### إنشاء وقراءة وتحديث ملف الـ SysColor.ini

ان آخر جزئية فى الـ SysColor code فى القائمة (١١-١٢) توضح كيف ينشئ البرنامج ويقرأ ويحدث ملف الـ SysColor.ini. التوضيح يعقب القائمة.

يقوم الـ LoadSettings Procedure بقراءة الـ SysColor.ini اذا وجد الملف. ان دراسة هذا الـ code سطرأ بسطر يزودك بمعلومات يمكنك استخدامها فى نظم التعامل مع بدء الملف الخاصة بك. لحمل مواصفات الملف، يقوم الـ

## الباب الحادي عشر : Navigating Directories and Files

LoadSettings بإنشاء object من ال TStringList وهو ال IniValueList . يستدعى البرنامج بعد ذلك ال ReadSectionValues لل TStringList object . وتشير argument 'SysColor' الى القطاع الذي يجب قراءته في ملف ال .ini ، وهو في هذه الحالة ، [SysColor] . كل عنصر في ال IniValueList الناتج يعتبر String في ال form :

Identifier=Value

القائمة (١١-٢) SysColor\Main.pas

```
{ Load colors and options from SysColor.ini if present }
procedure TMainForm.LoadSettings;
var
  IniValueList: TStringList;
  I: Integer;
begin
  IniValueList := TStringList.Create;
  try
    { [SysColor] settings }
    IniFile.ReadSectionValues('SysColor', IniValueList);
    for I := 0 to IniValueList.Count - 1 do
      with SysColorArray[I] do
        begin
          CurrentColor := StrToInt(
            IniValueList.Values[IniItemList[I]]);
          OriginalColor := CurrentColor;
        end;
      { [Options] settings }
      SaveCheckBox.Checked :=
        IniFile.ReadBool('Options', 'Save settings', False);
    finally
      IniValueList.Free;
    end;
    ChangeSystemColors;
    InitSysColorArray;
  end;
```



```
{ Create or update SysColor.ini color settings only }
procedure TMainForm.SaveSettings;
var
  I: Integer;
begin
  { [SysColor] settings }
  for I := 0 to IniItemList.Count - 1 do
    IniFile.WriteString('SysColor', IniItemList[I],
      IntToStr(SysColorArray[I].CurrentColor));
end;

{ Write check box setting to SysColor.ini }
procedure TMainForm.SaveCheckBoxClick(Sender: TObject);
begin
  IniFile.WriteBool('Options', 'Save settings',
    SaveCheckBox.Checked);
end;
```

للحصول على جزئية ال Value لكل صفه ، استخدم ال Values array فى قائمة ال string ال IniValueList . يستخدم ال LoadSettings هذه التقنية لتعيين قيم لون ال SysColorarray . اختبر هذا التعيين بدقة :

```
CurrentColor := StrToInt(
  IniValueList.Values[IniItemList[I]]);
```

ان التعبير IniItemList[I] يشير إلى واحدة من ال labels التى تم إنشاءها فى ال OnCreate الخاص بال form ، (انظر ال FormCreate procedure والعبارات التى تبدأ من IniItemList... with ). وهذه ال strings تمثل جزئية ال Identifier لتحديد ال Identifier=Value . استخدم ال Identifier string كفهرس فى ال Values array للحصول على ال Value الخاص بهذا التحديد . يعتبر هذا ايضاً string ، لذا الكى تعيينه لل CurrentColor ، يقوم البرنامج باستدعاء ال StrToInt .

يستخدم ال LoadSettings تقنية ملف ال ini . اخرى لقراءة مواصفات ال Save عند خروج ال CheckBox استدع ال ReadBool ل IniFile object ،

## الباب الحادي عشر: Navigating Directories and Files

مروراً بعنوان القطاع ('Options')، ويعرف المتغير بـ ('Save settings')، وقيمة افتراضية. إذا كان التحديد موجوداً، فإن الـ ReadBool يقوم بإدخاله. إذا لم يكن، فإنه يدخل القيمة الافتراضية، المعينة هنا لخاصية الـ Checked التابعة للـ SaveCheckBox. استدع الـ functions الـ ReadInteger والـ ReadString والمتشابهتين لقراءة قيم الـ string والعدد الصحيح.

بعد تحميل مواصفات ملف الـ ini، يقوم الـ LoadSettings بتحرير الـ IniValueList. يقوم الـ procedure باستدعاء الـ ChangeSystemColors لنقل الألوان التي تم تحميلها إلى الـ Windows. والاستدعاء الأخير للـ InitSysColorArray يعيد بدء الـ array بالحصول على تلك الألوان المحفوظة مرة أخرى من الـ Windows. هذا يضمن أن القيم الموجودة في الـ array هي التي تم استخدامها فعلاً - إنها قد تختلف في بعض الأحيان نتيجة الطريقة التي يختار الـ Windows بها الألوان المتجانسة بشكل وثيق اعتماداً على إمكانيات الـ display driver.

يعتبر الـ SaveSettings Procedure أبسط من الـ LoadSettings. تقوم الـ for loop باستدعاء الـ WriteString method الخاص بالـ IniFile لكتابة مواصفات اللون الحالي. قم بتحديد عنوان قطاع ملف الـ ini بـ ('SysColor')، وجزئية الـ Identifier لكل صفة بـ (InitItemLIst[I])، و string يمثل جزئية الـ Value إلى الـ WriteString. وهنا، يقوم الـ IntToStr بتحويل قيمة الـ CurrentColor من الـ SysColorArray. عندما تنتهي من كتابة كل الألوان، يقوم الـ SaveSettings بتحرير الـ IniFile object.

لاحظ أن الـ WriteString لا يقوم بحفظ خاصية الـ Checked للـ SaveCheckBox. عندما حاولت أن أفعل هذا بالضبط في نسخة سابقة، لاحظت أن حفظ هذا الـ CheckBox قد كونه حاله 22-catch. يمكنك فحص الـ box، الذي تم حفظه في ملف الـ ini، ولكن إذا لم تفحصه، لا يتم تحديث الملف، مما يؤدي إلى تحديث الملف مرة أخرى من الملف القديم عندما يبدأ الـ SysColor في المرة القادمة.

والحل هو أن يقوم بتحديث قيمة الـ SaveCheckBox في كل مرة يتغير فيها الـ CheckBox. يتولى الـ SaveCheckBoxClick Procedure هذه المهمة للـ

OnClick event الخاص بال control يقوم ال procedure باستدعاء ال WriteBool لحفظ الصفة الحالية (الذى يكون قد تغيرت بالفعل بمجرد استدعاء ال OnClick). هذا يشير الى قاعدة هامة فى استخدام ملفات ال ini. لحفظ خيارات برنامج- إنك قد تحتاج الى تحديث عناصر منفردة اذا كانت قيمهم تؤثر على اذا ما كان ال ini. قد تم تحديثه .

### اختبار ملف SysColor.ini

توضح القائمة (١١-١٣) ملف SysColor.ini. ارجع الى هذه القائمة اثناء فحص نظم معالجة ملف ال ini. الخاص بال SysColor.

ان باقى قائمة ال SysColor، والتي يمكن ان تجدها على القرص المدمج فى دليل ال Source\SysColor، يوفر ال event handler لأضرار النافذة وال CheckBox. وهذه ال event handler، والتي لها اهداف واضحة، تقوم باستدعاء ال procedures اخرى فى البرنامج، لذا لن اذكرها أو اشرحها هنا. إننى اضفت تعليقات الى النافذة، فى حالة ان كنت تريد فحص المزيد من البرنامج.

#### القائمة (١١-١٣): ملف ال SysColor.ini

```
[SysColor]
Scroll Bar=12632256
Background=4210688
Active caption=8755968
Inactive caption=8421440
Menu=12632256
Window=16777215
Window frame=0
Menu text=0
Window text=0
Caption text=16777215
Active border=4227072
Inactive border=8421440
App work space=12632256
Highlight=32768
Highlight text=16777215
```

## الباب الحادى عشر : Navigating Directories and Files

Button face=12632256  
Button shadow=8421504  
Gray text=8421504  
Button text=0

[Options]  
Save settings=0

### افكار للمستخدم الخبير

• اذا قمت بتخزين ملفات ini. فى مسارات محددة، تأكد من منح المستخدمين فرصة اختيار موضع بديل . فى اغلب الحالات، من الأفضل ان تخزن ملفات ال ini فى دليل Windows، ولكن هذا ليس طلب. لتخزين ملفات ال ini. فى دليل البرنامج المركب، استخدم function ParamStr(0) للحصول على مسار التطبيق.

• استدع ال DiskFree لتحديد حجم مساحة القرص المتاحة، والتي قد تريد إضافتها الى directory اختيار الملف الخاص بهذا الباب . اختبر functions والدليل والملف الاخرى بالبحث فى online help الخاصة بـ Delphi لـ "File-management routines".

• لا تندفع وراء تقنيات التعامل مع الرسالة الخاصة بهذا الباب، قبل كتابة ال message event handler لتطبيق أو form، افحص بدقة اذا ما كانت الإمكانية التي تحتاجها توجد بالفعل فى صورة event handler.

• فى ال C، يمكن ان تقوم عبارة بتعيين قيمة ذات علامة مثل ال 1- لـ unsignedWORD parameters، ان قواعد type-checking الخاصة بـ Pascal لا تسمح بهذا العمل، ولكن الغير المباشر أمر بسيط. على سبيل المثال، لتمرير ال 1- الى WORD parameter للـ Windows API Function استخدم الثابت العشري المساوى وهو \$FFFF.

• تذكر دائماً أن تقرير ال objects التي تنشئها. عند إنشاء objects ديناميكية عامة مثل ال TStringLists، قم بإنشاءها فى ال OnCreate الخاص بالـ

form . استدع Free لكل objects تم إنشاؤه فى ال OnDestroy الخاص بال form .

• لا تنشئ ملفات مثل ال system.ini أو ال win.ini ، اللذان لا يجب إنشاء ملفات مشابهة لهما ولقد تم استبدالها بال Windows registry .

### المبتدعات التى يمكنك تجربتها

(١١-١): اكتب ini file editor .

(١١-٢): اضع قطاعاً إلى ملف ال SysColor.ini ليحفظ وضع نافذة البرنامج . ان تشغيل ال SysColor يجب ان يعيد النافذة الى اخر وضع لها .

(١١-٣): اكتب wildcard filter editor خاص بك ، مشابهاً لذلك الخاص بـ Delphi ، واضفه الى directory dialog القابل للتغيير الخاص بهذا الباب . اكتب user filters لإنشاء ملف (.ini) .

(١١-٤): اكتب file utility يمكن ان تغير مميزات الملف مثل مواصفات ال Archive وال ReadOnly . استخدم ال function s . ال FileGetAttr وال FileSetAttr فى نظم إدارة الملف الخاصة بـ Delphi (ابحث عنهم فى ال online) .

(١١-٥): متقدم فى هذا الباب العديد من التقنيات اللازمة لإنشاء file Explorer utility مخصصة . هناك دائماً سوقاً جيدة للبرامج من هذه النوعية التى تدعم ادوات ال Windows . المعيارية . مشروع متقدم ، فكر فى كتابة البديل الخاص بك لـ Explorer utility الخاصة بال Windows . يمكنك مثلاً اضافة خاصية تحافظ على drives القرص المتعددة مفتوحة - شئ لا يستطيع ال Explorer ان يفعله .

## الباب الحادى عشر : Navigating Directories and Files

(٦-١١) : متقدم - يقوم الـ SysColor بتحديث كل ألوان نظام الـ Windows ، حتى إذا كانت قيم اللون لم تتغير . هذا ينتج عنه العديد من الرسائل التى تصدر صوتاً بين حواجز event التطبيق . أكتب نسخة أكثر فعالية تقوم بتحديث أقل عدد فقط من الألوان عندما يضغط المستخدمون زر الـ Set .

### ملخص:

- استخدم الـ directory components والملف الموضحة فى هذا الباب فقط . عندما تحتاج الـ File-Selection dialog مخصص . يوضح الباب التالى كيفية استخدام الـ components الـ SaveDialog والـ OpenFileDialog الخاصة بـ Delphi ، واللذان يعتبران كافيتين للـ directory interface objects والملف .
- استخدم الـ DirectoryListBox ، والـ DriveComboBox ، والـ FileListBox ، والـ FilterComboBox components . لإنشاء الـ File-Selection dialogs مخصصة . بربط هذه الـ objects ، فإنك تمكنها من العمل مع بعضها بحيث ، مثلاً ، عندما يقوم المستخدمون بتغيير الـ directories فى الـ DirectoryListBox ، يقوم الـ FileListBox المرتبطة به بصورة تلقائية بتحديث محتوياته .
- اضعف الـ ShellAPI لأمر الـ uses الخاص بالـ module . واستدع الـ ShellExecute لتشغيل برامج أو تحميل registered files . على سبيل المثال ، يمكنك عرض ملف نص فى الـ Windows Notepad utility بفتح ملف .txt . بالـ ShellExecute .
- ان الـ Delphi components لا تميز أسماء ملفات السحب والاسقاط ، ولكن يمكنك بسهولة إضافة هذه الامكانية بكتابة procedure يستجيب لرسالة الـ wm\_DropFiles . لتشغيل ملفات السحب والاسقاط ، استدع الـ DragAcceptFiles فى الـ OnCreate بالـ form .

● عندما يحدث minimize لتطبيق، تختفى الform الرئيسية. لهذا السبب، فإن تنفيذ ملفات السحب والاسقاط لبرنامج minimized مثل ايقونة مما يتطلب من الApplication object ان يستجيب لرسالة الwm\_DropFiles. لبرمجة استجابة لهذه الرسالة أو لأى من رسائل الWindows الأخرى، قم بتعيين message handler لخاصية الOnMessage التابعة للApplication. يجب ان يؤدى هذا التعيين فى وقت التشغيل.

● تقوم ملفات الInitialization بتخزين خيارات ومواصفات وقيم أخرى. استخدم الTIniFile class لإنشاء object يمكنك ان تستدعى له methods مثل الReadSectionValues، والReadBool، والWriteString لقراءة وكتابة مواصفات ملف ال.ini.

● توضح الSysColor utility الخاصة بهذا الباب الmethods العملية للتعامل مع ملف ال.ini. ويوضح البرنامج ايضاً تقنيات عديدة ومفاهيم برمجة Pascal.

يغطى الباب التالى Delphi dialog components، ويظهر افكاراً لكتابة الmodal و modeless dialog windows.

\*\*\*\*\*

# الباب الثاني عشر

## التعامل من خلال Dialog Boxes

محتويات هذا الباب:

- Components
- Dialog modes
- Common dialogs
- Paged dialogs
- البحث من خلال الـ dialogs
- Page controls
- تحديد حجم النافذة
- انشاء docking controls

في الـ Windows، يعتبر الـ Dialog Box من الناحية الفنية نافذة ذات طرازاً محدداً تفتقد ازرار التكبير والتصغير، وغالباً من تحتوى على check boxes، radio buttons، و Conrtols اخرى. ولكن في Delphi، أى form يمكن ان يكون لها سمات الـ dialog box، والخطوط الفاصلة بين dialogs التطبيق ونوافذه الأخرى تعتبر غير واضحة في ضوء واجهة التطبيق الجرافيكية الحديثة للمستخدم.

وقد تعتقد ان الـ dialog box، كما اعنى هنا، هو اذن وصوت البرنامج. إنه يمنح المستخدمين وسيلة لإختيار الخيارات في إدخال المدخلات. ويمكن للـ dialog



box أن يرسل رسائل، يسأل أسئلة، ويتلقى إجابات. في هذا الباب، سوف تتعرف على الـ dialog components الخاصة بـ Delphi، وكذلك الـ TabControl، الـ PageControl، والـ TabSheet، والتي جعلت من السهل إنشاء dialogs ونوافذ أخرى مع الـ page-tab controls في مثل page tabs في الـ Delphi VCL palette، و dialog boxes متعددة الصفحات. سوف أوضح أيضاً في هذا الباب إثنين من تقنيات البرمجة الحديثة في Delphi - كيفية إنشاء docking windows، وكيفية إضافة قيود على أحجام النوافذ.

وليكون مرجعاً كاملاً، يوضح هذا الباب أيضاً كيفية استخدام الـ Notebook، والـ TabbedNotebook، والـ TabSet الخاصة بالـ Win 3.1 القديم. يجب على التطبيقات الجديدة أن تستخدم الـ TabControl، والـ PageControl، والـ TabSheet ولكنك تحتاج معلومات عن هذه الـ components القديمة إذا كان تطبيقك يجب أن يتم تشغيله في الـ Windows 3.1 أو إذا كنت تطور برمجيات قديمة لتستخدم components حديثة.

## :Components

فيما يلي قائمة الـ dialog components العام الخاص بـ Delphi :

● **ColorDialog**: إن هذا الـ component يتضمن dialog اللون للـ Windows. استخدمه كأداة لاختيار اللون. Dialogs : Palette.

● **FindDialog**: يتضمن الـ search dialog للـ Windows. استخدمه لأداء بحث للبيانات. Dialogs : Palette.

● **FontDialog**: يتضمن الـ font-selection dialog للـ Windows. استخدمه للسماح للمستخدمين باختيار text Font، ولتحديد لون الواجهة للنص المعروض. Dialogs : Palette.

● **Notebook**: يقدم هذا الـ component حاوية متعددة الصفحات والتي يمكنك استخدامها مع الـ TabSet لتوفير toolbars مجدولة ومجموعة controls أخرى. Win3.1 : Palette.

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

• **OpenDialog**: ويحتوى Open file dialog الخاص بالـ Windows .  
استخدمه لإختيار الملفات وتصفح الأدلة، كاستجابة نموذجية لأمر الـ FileOpen .  
Dialogs : Palette .

• **PageControl**: يمكنك إنشاء page-tab بأقل برمجة مطلوبة . كل page في الـ PageControl يعتبر ، TabSheet object منفصل . وعلى هذه الـ objects ، يمكنك إسقاط controls أخرى مثل الأزرار ، check boxes ، radio-boxes - أى شئ تريده . يستطيع المستخدمون ضغط أبواب الـ PageControl page tabs للإختيار من بين صفحاتها والوصول للـ control الموجود على كل page . إن طريقة رائعة لوضع العديد من الـ controls على مساحة صغيرة- على سبيل المثال ، فى برنامج ذى اختيارات كثيرة . Win32 : Palette .

• **ReplaceDialog**: يتضمن الـ replace dialog الخاص بالـ Windows .  
إستخدمه لأداء عمليات البحث والابدال . Dialogs : Palette .

• **SaveDialog**: يتضمن الـ dialogSave-fil الخاص بالـ Windows .  
استخدمه لإدخال أو إختيار اسماء ملف ، لتصفح الأدلة ، كاستجابة لأمر الـ FileSave . Dialogs : Palette .

• **TabbedNotebook**: يقدم هذا الـ component الـ multipage panel-like object والذي لها ابواب ذات labels على الحافة العلوية يستطيع المستخدمون اختيار صفحات الـ TabbedNotebook بإختيار الـ labels ، والتي تجعل هذا الـ component ملائم بشكل خاص للـ dialogs الخيارات المعقدة ذات الـ controls العدد الكثير من بحيث ان تناسب داخل نافذة واحدة . Win3.1 : Palette .

• **TabControl**: تنشئ dialogs متعددة الصفحات ولكن ، الـ TabControl يعتبر object واحد- ان "صفحاته" هى مجرد خداع ، وهى المسئولة عن التحكم فيها فى وقت التشغيل . يمكنك استخدام الـ TabControl مع مجموعة أخرى من الـ controls التى لها محتويات مختلفة- سلسلة من حقول الـ editing مثلاً- إعتماًداً على الـ page-tab المختار . وهذا يختلف عن الـ PageControl والذي تكون فيه كل صفحة TabSheet منفصل يمكن ان يحمل مجموعة من الازرار و الـ controls الاخرى . Win32 : Palette .

• **TabSet**: يقدم ال toolbar والذي يتكون من عدد من tabs ، والتي يمكنك استخدامها بالارتباط مع components اخرى لتزودهم بإمكانية الانتقال بين ال pages . يوضح ال TabEdit الخاص بهذا الباب كيفية استخدام ال TabSet لإنشاء text editor متعدد الصفحات . Palette : Win3.1 .

• **TabSheet**: يقوم ال PageControl بتخزين واحداً أو أكثر من ال TabSheet objects ، والذي يحمل ال controls التي تظهر على كل page من النافذة أو ال dialog متعدد ال pages . لإنشاء ال TabSheet ، اضغط على الزر الأيمن للفأرة داخل ال PageControl واختر أمر ال New Page من القائمة . None : Palette

## : Dialog Modes

ان التطبيقات تستخدم dialogs بوحدة من طريقتين : كنوافذ ال modal تحتفظ بال input focus حتى الإغلاق ، أو كنوافذ modeless تسمح للمستخدمين بالتحويل بعيداً عنها . على سبيل المثال ، يعتبر dialog الاختيارات نافذة modal يجب على المستخدمين إغلاقها قبل الاستمرار فى استخدام التطبيق . ولكن ، dialogs البحث والإبدال يعتبر modeless بحيث يستطيع المستخدمون بدء البحث ، التحويل الى نافذة dialog ، ثم استكمال البحث .

## : Modal dialogs

استدع ال ShowModal method الخاص بال form لعرض ال dialog . يجب على المستخدمين ان يغلّقوا نافذة ال dialog لاستكمال البرنامج . على سبيل المثال ، اذا كان ال MyDialog يعد form ، فإن أمر قائمة أو onClick لزر ما قد يعرض نافذة ال dialog بالعبرة :

MyDialog.ShowModal;

فى المعتاد ، تقوم ازرار ال Close ، وال Cancel ، أو ازرار إغلاق ال form الاخرى الخاصة بال dialog بتعيين قيمة لل ModalResult ، الذى يقوم ال ShowModal بإدخاله . لتحديد ما اذا كان المستخدمين قد ضغطوا زر ال OK لل dialog ، استخدم عبارة مثل التالية لاختيار قيمة إدخال ال ShowModal :

## الباب الثاني عشر : التعامل من خلال Dialog Boxes

```
if MyDialog.ShowDialog = mrOk then  
{ ... take action if user clicked OK button }
```

\* في dialog modules، قم بتعيين القيمة التي تريد للـ ShowModal ان يدخلها للـ ModalResult. إما ان تحدد mork لخاصية الـ ModalResult أو في الـ OnClick الخاص بزر الـ OK للـ OnClick استخدم عبارة مثل هذه :

```
ModalResult := mrOk;
```

**تحذير:** القيم التي تم تعيينها للـ ModalResult يجب ان تكون غير صفرية حتى يعود الـ ShowModal.



### عرض انواع Objects اخرى من Modal Dialog

يقوم الـ ShowModal بادخال قيمة عدد صحيح لإدخال من object انواع اخرى من modal dialog أضف هذه function إلى الـ form class لإدخال البيانات التي تحتاجها. على سبيل المثال، قم بتعريف String function في القطاع الـ public للـ form class

```
TYourForm = class(TForm)
```

```
...
```

```
public
```

```
function GetStringResult: String;
```

```
end;
```

نضن الـ function لإرجاع نتيجة الـ dialog مثل خاصية الـ Text

التابعة للـ Edit،

```
function TYourForm.GetStringResult: String;
```

```
begin
```

```
with Edit1 do
```

```
if Length(Text) = 0 then
```

```
Result := 'Default string'
```

```
else
```

```
Result := Text;
```

```
end;
```

استدع الـ ShowModal لعرض نافذة dialog form ثم استدع

function الـ "get result" مباشرة بعدها،

```
if YourForm.ShowDialog = mrOk then
```

```
S := YourForm.GetStringResult;
```

## : Modeless dialogs

كما يعرفه ال Windows ، ال modeless dialog يعتبر Child Window لا تملك input focus وتسمح للمستخدمين بتشغيل التطبيق بشكل طبيعي اثناء عرض ال dialog box على الشاشة. ولكن في Delphi ، لعرض form على انها modeless dialog ، استدع ال Show method :

```
MyDialog.Show;
```

ان استدعاء ال Show يحدد خاصية ال Visible لل form ب True . ويستدعي ال BringToFront لعرض ال dialog box فوق النوافذ الاخر- . لإخفاء نافذة ال dialog ، حدد ال Visible ب False :

```
MyDialog.Visible := False;
```

لتحدد ما اذا كان modeless dialog نشط أم لا ، افحص خاصية ال Visible له .

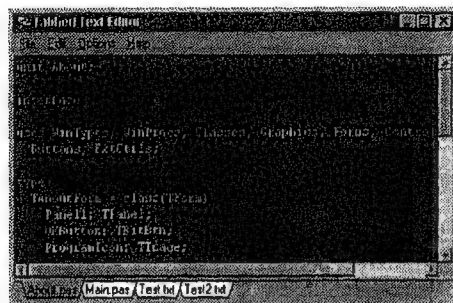
## : Common Dialogs

**على القرص المدمج:** ان اربعة من ال common dialogs - وهي ال OpenFileDialog ، ColorDialog ، FontDialog ، SaveDialog - توفر نوافذ اختيار ، ولون ، واسم ملف . يوضح تطبيق ال TabEdit على القرص المدمج كيفية استخدام هذه ال components ، ان ال source code لل TabEdit طويلة جداً بحيث يصعب ذكرها هنا كاملة ، لذا قدمت القوائم في مقتطفات صغيرة نسبياً . بالطبع ، توجد ال source code كلها على القرص المدمج في دليل ال Source\TabEdit . يوضح الشكل (١٢-١) عرض ال TabEdit . قم بتحميل ملف مشروع ال TabEdit في Delphi لفحص البرنامج كاملاً .



**ملحوظة:** يستخدم ال TabEdit ال Win3.1 components لإنشاء صفحاته المتعددة ، التي يتم اختيارها بامتياز ال page tabs في اسفل النافذة . مؤخراً في هذا الباب ، سوف اوضح كيفية استخدام ال components TabControl ، TabSheet ، وال PageControl الحديثة لإنشاء dialogs ونوافذ متعددة الصفحات .

Note



شكل (١٢-١) : يقدم الـ TabEdit text editor متعدد الصفحات، مشابهاً للـ code editor الخاص بـ Delphi

### : Font and Color dialogs

كما توضح القائمة (١٢-١)، ان components الـ ColorDialog والـ FontDialog سهلة الاستخدام. أضف هذه الـ objects على form استبدع الـ Execute methods الخاصة بها لعرضها كـ modal dialogs. اذا عاود الـ Execute بـ True، فقد ضغط المستخدم الـ OK لإغلاق نافذة الـ dialog، ويمكنك نسخ أو استخدام خاصية الـ Font للـ FontDialog أو خاصية الـ Color للـ ColorDialog للحصول على اختيار المستخدم.

القائمة (١٢-١)، ان components الـ ColorDialog والـ FontDialog يسهل استخدامها

```
procedure TMainForm.OptionsFontClick(Sender: TObject);
begin
    if FontDialog1.Execute then
        Memo1.Font := FontDialog1.Font;
end;
```

```
procedure TMainForm.OptionsBackgroundClick(Sender: TObject);
begin
    if ColorDialog1.Execute then
        Memo1.Color := ColorDialog1.Color;
end;
```

يوفر ال `ColorDialog` ثلاث اختيارات، والتي يمكنك برمجتها باستخدام ال `Object Inspector` أو بتعيين `True` أو `False` لثلاث خصائص أثناء التشغيل:

● **cdFullOpen**: عندما يكون هذا الاختيار محدد بـ `True`، يوفر ال `dialogs`، `grid` لاختيار لون والتي يمكن للمستخدمين ان ينتقوا الالوان منها. عندما يكون ال `cdFullOpen` محدد بـ `True`، يظهر ال `dialog` اللون لوحة الاختيار بها `controls` قيم اللون الاحمر والاخضر والازر وقيم الالوان الاخرى. على الحاسبات البطيئة نسبياً، قد تمر عدة ثوان قبل ان ينشئ ال `Windows` ال `dialogs`، فلا تستخدم هذا الاختيار اذا كنت تعلم ان تطبيقك يحتاج الى تشغيل على نظم بطيئة. مازال المستخدمون يفتحوا ال `dialog` بضغط زر الالوان المخصص لذلك.

● **cdPreventFullOpen**: محدد بـ `True` لمنع المستخدمين من فتح `dialog` اختيار اللون (يكون زر الالوان المخصص دائماً بـ `disabled`). يمكنك استخدام هذا الخيار لقصر الالوان على مجموعة سابقة التحديد.

● **cdShowHelp**: حده بـ `True` لعرض زر ال `Help`. محدد بـ `False` اذا كان برنامجك لا ينفذ `online help`. اذا كان لدى برنامجك `online help`، حدد ال `cdShowHelp` بـ `True` وعين قيمة مناسبة لخاصية ال `HelpContext` التابعة للـ `dialog`.

ويوفر ال `FontDialog` خيارات عديدة، بعضها واضح-مثل ال `fdTrueTypeOnly` وال `fdFixedPitchOnly`. فيمايلي بعض الافكار عن الخيارات الأقل وضوحاً:

● **fdAnsiOnly**: حده بـ `True` لعرض ال `font` التي تنفذ مجموعة رموز ال `Windows` فقط على سبيل المثال، عندما يكون ال `fdAnsiOnly` محدد بـ `True`، لا يقوم ال `font dialog` بذكر ال `WingDings Symbol font`.

● **fdEffects**: حده بـ `False` لابطال خيارات اللون، `underline`، `strikeout`. يجب ان تقوم دائماً بإيقاف عمل هذه العناصر اذا كان برنامجك لا يستخدمها. ولكن للاسف لا يمكن تشغيل وإبطال الخيارات الثلاثة اختياريّاً (تحديد من ال `Windows`). هذا الخيار للكل أو لا شيء.

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

● **fdNoFaceSel**: من الطبيعي، يعرض الـ font dialog الحال في جزئية الـ edit للـ combo Box الخاص بقائمة الـ font. حدد الـ fdNoFaceSel بـ True إذا كنت تريد أن يكون الـ edit control خالياً عندما يظهر الـ dialog.

● **fdNoSimulations**: حدده بـ True لمنع ذكر الـ font الشبيهه بالـ GDI في قائمة. وهذا المفتاح ليس له تأثير على الـ fonts الشاشة؛ فقط لـ fonts الطباعة.

● **fdNoStyleSel, fdNoSizeSel**: هذه المفاتيح تحدد ما إذا كانت اختيارات النمط والحجم قد تم اختيارها بداية (تم إبرازها). إذا كنت تريد أن يختار المستخدمون font جديد، ولكنك لا تريد عرض سمات الـ font الحال، حدد هذان الاختياران والـ fdNoFaceSel بـ True.

● **fdWysiwyg**: حدده بـ True لذكر فقط الـ font المتاحة لكل من الطباعة والشاشة. عندما يكون الـ fdWysiwyg محدد بـ False، يستطيع المستخدمون اختيار fonts قد لا تطبع كما هي معروضة. إذا كان النص يظهر فقط على الشاشة ولا يطبع، حدد هذا الاختيار بـ False.

● **fdLimitSize**: حدده بـ True فقط عندما تحدد قيم خاصية الـ MinFontSize والـ MaxFontSize لقصر اختيارات الحجم على مدى محدد.

**ملحوظة:** يمكنك أيضاً استخدام الـ FontDialog لاختيار fonts مخصصة، أو متجانسة مع طباعة معينة.

Note

### Open and Save dialogs

بوصفه text editor، يعتبر الـ TabEdit برنامجاً مفيداً في مجاله، ولكنه يعمل أيضاً كـ template لأي برنامج يتعامل مع ملف ان الخطوة الأولى هي كتابة procedures، واحداً لقراءة ملف من على القرص، والثاني لإنشاء أو كتابة بيانات ملف. إذن اكتب الـ procedures منفصلين دائماً، بدلاً من اخراج وادخال ملف الـ code في الـ event handlers لأمر القائمة هذا يجعل الـ procedure ملف البرنامج متاحة لأي نظم تحتاجها. توضح القائمة (١٢-٢) الـ SaveFile والـ LoadFile للـ TabEdit.



ان ال LoadFile و ال SaveFile يستخدم ال Pascal exceptions لتجربة عبارات قد تفشل . فى حالة الاخطاء ، تقوم عبارة except بعرض رسالة ال display. يوضح الباب التاسع عشر المزيد حول التعامل مع exceptions ، ولكنه سهل الاستخدام . فقط ضع كل العبارات التى قد تولد exceptions- التابعة لل procedure فى ال except ، وقم بإنهائه بكلمة ال end الرئيسية . ان جزئية ال on-do فى ال except . تمسك بـ except objects معينة (ال EReadError و ال EWriteError فى هذه الحالة) . من المهم ان تمسك فقط بال exceptions التى تريد التعامل معها ، وتسمح للآخرين بالتعامل مع ال exception handlers ال افتراضية فى Delphi .

#### القائمة (١٢-٢) Tab Edit's file input and output procedures:

```
{ Read file from disk }
procedure TMainForm.LoadFile(const Path: String);
begin
  with Pages[TabSet1.TabIndex] do
    try
      Memo1.Lines.LoadFromFile(Path);
      Dirty := False;
      Page.Clear;
      SetFilename(Path);
    except on e: EReadError do
      MessageDlg('Error reading file', mtError, [mbOk], 0);
    end;
  end;
end;

{ Write current file to disk }
procedure TMainForm.SaveFile(Index: Integer);
begin
  with TabSet1, Pages[Index] do
    begin
      try
        Memo1.Lines.SaveToFile(Filename);
        Dirty := False;
      except on e:EWriteError do
```

## الباب الثاني عشر : التعامل من خلال Dialog Boxes

```
MessageDlg('Error writing file', mtError, [mbOk], 0);
end;
```

```
end;
```

```
end;
```

توضح القائمة (١٢-٢) event handlers لقائمة File لـ TabEdit .  
تستخدم الـ procedures الـ SaveDialog components والـ OpenFileDialog ،  
وتستدعى الـ procedures الموجودة في القائمة (١٢-٢) . استخدم هذه القائمة  
كممرشد لك في إضافة أوامر الـ Open والـ Save والـ Save As لتطبيقاتك .

القائمة (١٢-٢) : الـ event handlers لقائمة File لـ TabEdit

```
{ File|Open command }
procedure TMainForm.FileOpenClick(Sender: TObject);
begin
  with Pages[TabSet1.TabIndex] do
    begin
      if Dirty then FileSaveClick(Sender);
      if {still} Dirty then Exit; { File not saved }
      if FileOpenDialog.Execute then
        LoadFile(FileOpenDialog.Filename);
    end;
  end;
end;
```

```
{ File|Close command }
procedure TMainForm.FileCloseClick(Sender: TObject);
var
  W: Word;
begin
  with TabSet1, Pages[TabIndex] do
    begin
      if Dirty then
        begin
          W := MessageDlg(
            'Save changes to ' + Tabs[TabIndex] + '?',
            mtWarning, [mbYes, mbNo, mbCancel], 0);
```

```

case W of
    mrYes: FileSaveClick(Sender);
    mrNo: Dirty := False;
    mrCancel: Exit;
end;
end;
if {still} Dirty then Exit; { File not saved }
Page.Clear;
Memo1.Clear;
Filename := untitledName;
Tabs[TabIndex] := Filename;
end;
end;

{ File|Save command }
procedure TMainForm.FileSaveClick(Sender: TObject);
begin
    with TabSet1, Pages[TabIndex] do
        if Filename = untitledName then
            FileSaveAsClick(Sender)
        else
            SaveFile(TabIndex);
end;

{ File|Save As command }
procedure TMainForm.FileSaveAsClick(Sender: TObject);
begin
    with TabSet1, Pages[TabIndex] do
        if FileSaveDialog.Execute then
            begin
                SetFilename(FileSaveDialog.Filename);
                SaveFile(TabIndex);
            end;
end;
end;

```

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

ان الـ `FileOpenClick` يحفظ الملف الحالي اذا كان الـ `Dirty` محدد بـ `True`، مما يشير الى ان المستخدم قد قام بتغييرات للملف. بعد هذا العمل، يعرض الـ `FileDialog` الـ `procedure` باستدعاء الـ `Execute method` الخاص به. اذا حدد الـ `method` بـ `True` يقوم الـ `LoadFile` عندئذ بقراءة الـ `Filename` المختار التابع للـ `dialog`.

يحث الـ `FileCloseClick` المستخدمين على حفظ اية تغييرات وإغلاق الملف الحالي. ان الـ `FileSaveClick` يتحول الى الـ `FileSaveAsClick` اذا كان الملف الحالي لا يحمل اسماً: ولا يستدعى الـ `SaveFile` الـ `procedure`. لمحاولة كتابة الملف الحالي على قرص. لا يستعمل أى من هذين الـ `procedure` أى `dialog components`.

يقوم الـ `FileSaveAsClick` باستدعاء الـ `Execute` لـ `FileDialog` `object`. اذا حدد الـ `method` بـ `True`، يكون المستخدمون قد اختاروا اسم ملف جديد (أو سمحوا بالكتابة على ملف موجود)، ويستدعى البرنامج الـ `SaveFile` لكتابة الملف الحالي على قرص. إن عبارة الـ `SetFilename` تعين اسم الملف لـ `pagetab label`.

### أنواع الملفات المتعددة و Filters:

فى برنامج يفتح أنواع مختلفة من الملفات، من الأفضل دائماً أن تقتصر على الـ `SaveDialog filters` على نفس نوع الملفات الأحدث الذى تم فتحه. على سبيل المثال، اذا فتح المستخدم ملف نص، يمكن أن يعرض الـ `SaveDialog` ملفات الـ `txt`. فقط. اذا فتح المستخدم `bitmap`، سوف يعرض الـ `SaveDialog` ملفات الـ `bmp`. هذا يذكر المستخدم بنوع الملفات التى يفتحها ويحفظها. اذا قمت بدلاً من ذلك ببرمجة نفس الـ `filters` فى كلاً من الـ `OpenDialog` والـ `SaveDialog`، قد يتخيل المستخدم أنه بإمكانه التحول بين أنواع الملفات ببساطة بواسطة تغيير الإمتداد (على سبيل المثال، بحفظ ملف الـ `txt` على أنه ملف `bmp`). اذا لم يكن بإستطاعة تطبيقك أن يؤدي مثل هذه `filters`، فمن الأفضل أن تعتقد أنه يستطيع.

لإعادة برمجة الـ `SaveDialog Filter`، استخدم العبارة التالية بعد تنفيذ الـ `OpenDialog object`، عندما يحاول المستخدم فى المرة التالية أن يحفظ الملف،

يعرض ال SaveDialog فقط الملفات التى لها نفس الإمتداد . على سبيل المثال ، اذا فتحت ملف bmp ، سيكون ال string الناتج المعين لخاصية ال Filter ل SaveDialog سيكون Files \*.bmp(\*.bmp):

```
SaveDialog1.Filter :=
  'Files (*'
    + ExtractFileExt(OpenDialog1.FileName)
    + ')*'
    + ExtractFileExt(OpenDialog1.FileName);
```

**فكرة:** باستخدام ال Format function ال arguments المكررة فى الأقواس المربعة ، يمكن تقصير العبارة السابقة بحذف الإستدعاء المزدوج لل ExtractFileExt . على سبيل المثال ، هذه العبارة مساوية لل code

السابقة :

```
SaveDialog1.Filter := Format(
  'Files (*. %s)*. %0:s',
  [ExtractFileExt(OpenDialog1.FileName)]);
```

### إنشاء قائمة تاريخ:

ان ال OpenDialog وال SaveDialog لها خاصية TStrings ، وهى ال HistoryList ، التى يمكنك استخدامها لحفظ اسماء الملف التى تم اختيارها حديثاً . هذا ينشئ تاريخاً من اسماء الملفات التى يمكن للمستخدم ان يختار منها الملفات التى عمل بها سابقاً . يمكنك حتى حفظ قائمة التاريخ فى ملف لإستعادته فى المرة التالية لبدء البرنامج .

ان إنشاء قائمة تاريخ يتطلب ثلاث خطوات :

- ١ - أضف ال SaveDialog أو ال OpenDialog على from .
- ٢ - قم بتغيير خاصية ال FileEditStyle لل SaveDialog أو ال OpenDialog لتصبح fsComboBox .
- ٣ - أضف اسماء الملفات المختارة لخاصية ال HistoryList لل SaveDialog أو ال OpenDialog .

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

لإتمام الخطوة رقم ٣ ، فى OnClick أو button event handler والى تفتح أو نضغط الملف ، قم بتنفيذ الـ dialog واضف اسم ملف مختار لقائمة التاريخ بـ code مثل هذه :

```
with OpenFileDialog1 do
if Execute then
begin
{ Open file here }
FileName := Lowercase(FileName);
HistoryList.Add(FileName);
end;
```

يمكنك فعل نفس الشئ بالـ SaveDialog . عندما يقوم المستخدم بفتح الـ dialog فى المرة التالية ، تظهر الملفات الاحداث اختياراً فى نافذة الـ combobox edit . (جرب الـ code السابقة فى OnClick الخاص بالزر لترى التأثير تأكد من تحديد خاصية الـ FileEditStyle للـ dialog بـ fsComboBox ، أولاً تظهر قائمة التاريخ) .

لتحديد عدد اسماء الملفات فى الـ HistoryList ، قم بتعريف ثابت فى الـ module :  
const  
maxHistoryList = 6;

ثم ، استخدم البرمجة التالية لإدخال اسماء ملف فى الـ HistoryList .  
تظهر الملفات الاحداث اختياراً على رأس القائمة ؛ والملفات القديمة تتحرك الى اسفل عندما تصل القائمة الى اقصى حجم لها :

```
with OpenFileDialog1 do
if Execute then
begin
{ Open file here }
FileName := Lowercase(FileName);
with HistoryList do
begin
if Count = maxHistoryList then
Delete(Count - 1);
HistoryList.Insert(0, FileName);
end;
end;
```

قد تريد ايضاً ان تتضمن امراً أو اختياراً لمسح قائمة التاريخ . لفعل هذا، استخدم عبارات مثل هذه :

```
OpenDialog1.HistoryList.Clear;
SaveDialog1.HistoryList.Clear;
```

## : Paged Dialogs

بالرغم من ان الشاشات اليوم ذات تقنية عالية ، الا ان المساحة المتوفرة على شاشة تعتبر قليلة ونادرة . لتجد بعض المساحات الإضافية لـ controls و objects اخرى ، يمكنك إنشاء dialog boxes متعددة الصفحات . هذا يساعد ايضاً على تنظيم controls فى فئات يختارها المستخدمون باختيار tabs ذات labeled . تقترح الفصول التالية سبل إنشاء نوافذ بـ components الـ TabSet ، الـ Notebook ، والـ TabbedNotebook .

**ملحوظة:** الـ components الموضحة فى هذا الفصل توجد على الـ Win3.1 palette ، وهى متاحة لبرامج الـ Windows ذات الـ ٣٢ والـ ١٦ بت . البرامج الاحداث يجب ان تستخدم الـ PageControl والـ TabShee والـ TabControl الموضحة بعد ذلك فى هذا الباب . ولكن المعلومات المذكورة هنا عن الـ Win3.1 components مازالت ذات قيمة اذا كنت تحتاج ان تدعم الـ Win3.1 ، أو اذا كنت تقوم بتطوير تطبيقاً جديداً لتستخدم الـ controls الحديثة .

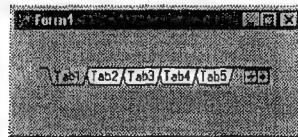
## ١. TabSet :

يوضح شكل (١٢-٢) الـ TabSet object فى نافذة اختيارية لإنشاء الـ object ، قمت بضغط الزر البيضاوى لخاصية الـ Tabs ، وادخل سبعة labels ، Tab1 ، Tab2 ، ... ، Tab7 . ولكن ، بسبب انه لا يوجد الا اماكن لستة labels من السبع ، يعرض الـ component ازرار اسهم افقية يمكن للمستخدم ضغطها لتحريك الـ TabSet يمينا ويساراً .

يستخدم تطبيق الـ TabEdit الـ TabSet object ، محاذاً للحافة السفلية النافذة ، لإنشاء الـ Memo . المتعدد الصفحات . فى الواقع ، ان للبرنامج Memo

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

object واحد. لتخزين ملفات متعددة في ذاكرة، يستخدم البرنامج الـ Pages array العام لسجلات الـ TPageRec. يحتوى كل سجل TPageRec على (Page) object الـ TStringList. عندما تختار tab لتقلب الصفحة، يعين البرنامج قائمة الـ string المختارة لـ Memo object.



شكل (١٢-٢): يعرض الـ TabSet أزرار اسهم أفقية بحيث يمكن للمستخدمين تحريك الـ Tabs يساراً ويميناً إذا لم يكن هناك مساحة كافية لكل الـ Tabs

توضح القائمة (١٢-٤) الـ TabSet event handlers التابعة للبرنامج. في أغلب الحالات، إنك تحتاج الى ان تستجيب لـ events فقط - واحداً تم توليده قبل تغيير الـ Tab واحداً تم توليده بعد ذلك. ان الـ two events يعطيان برنامجك الفرصة لفصل الـ page الحالية عندما يكون الـ Tab على وشك ان يتغير ثم يلحق صفحة جديدة بعد التغيير. ان معنى الصفحة يعتمد كلية على تطبيقك - ان الـ TabSet يعطيك وسائل منضبطة لإنشاء الخدع متعدد الصفحات.

القائمة (١٢-٤): الـ event handlers التابعان لـ TabSet في تطبيق الـ TabEdit

```
{ A tab is changing. Save Memo's text in a TStringList object }
procedure TMainForm.TabSet1Change(Sender: TObject; NewTab:
Integer;
var AllowChange: Boolean);
begin
  with TabSet1, Pages[TabIndex] do
    begin
      Page.Clear;
      Page.Assign(Memo1.Lines);
    end;
end;
```

```
{ A tab has changed. Assign a TStringList object to Memo }
procedure TMainForm.TabSet1Click(Sender: TObject);
```



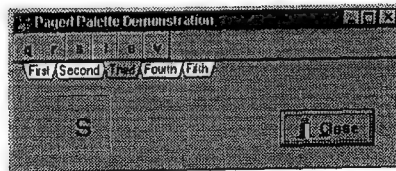
```
begin
  with TabSet1 do
    Memo1.Lines.Assign(Pages[TabIndex].Page);
end;
```

يقوم الـ TabSet1Change Procedure أولاً بمسح الـ TStringList الـ Page الحالي . بعد ذلك ، يستدعى الـ Procedure الـ assign لكل Page ، مروراً بخاصية الـ Lines للـ Memo (object من نوع الـ TStrings) . هذا يحفظ المحتويات الحالية للـ Memo فى الـ Page قبل ان يتغير الـ TabSet1.

بعد التغيير ، يقوم الـ TabSet1Click method مرة اخرى باستدعاء الـ Assign . ولكن هذه المرة ، يذهب التعيين فى الاتجاه العكسى - من Page الـ TStringList المفهرسة بالـ TabIndex لقائمة الـ Lines التابعة للـ Memo . الآن ، يعرض الـ Memo الصفحة الجديدة من النص المرتبط بالـ tab المختار .

## الـ Notebook :

على القرص المدمج ، يقدم الـ Notebook حاوية متعددة الصفحات التى تستخدم فى اغلب الاوقات مرتبطة بالـ TabSet ، وللحصول على مثال عملى حول هذه التقنية ، جرب تطبيق الـ Palette على القرص المدمج فى دليل الـ Source\Palette . مع اثنين من الـ event handlers فقط ، ينشئ البرنامج toolbar ، يشبه الـ VCL palette الخاصة بـ Delphi يوضح شكل (١٢-٣) مظهر الـ palette . اضغط page tab للانتقال الى صفحة اخرى من الـ SpeedButtons . اختر أى SpeedButtons لعرض الـ caption الخاص به فى الـ Label . توضح القائمة (١٢-٥) الـ source code للبرنامج .



شكل (١٢-٣): يوضح تطبيق الـ palette كيفية إنشاء page tab ، يشبه الـ VCL palette التابعة بـ Delphi

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

لاختصار المساحة في هذا الباب ، قمت بحذف اغلب تعريفات الـ SpeedButtons . الـ ٣٦ من الـ TMainForm class .

القائمة (١٢-٥) : Palette\Main.pas

unit Main;

interface

uses

SysUtils, Windows, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, Buttons, ExtCtrls, Tabs,  
StdCtrls;

type

TMainForm = class(TForm)

TabSet1: TTabSet;

Notebook1: TNotebook;

SpeedButton1: TSpeedButton;

SpeedButton2: TSpeedButton;

{ ... }

SpeedButton36: TSpeedButton;

BitBtn1: TBitBtn;

Label1: TLabel;

Bevel1: TBevel;

procedure TabSet1Click(Sender: TObject);

procedure SpeedButton1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

MainForm: TMainForm;

implementation

```
{ $R *.DFM }
```

```
procedure TMainForm.TabSet1Click(Sender: TObject);
begin
  with TabSet1 do
    Notebook1.ActivePage := Tabs[TabIndex];
end;
```

```
procedure TMainForm.SpeedButton1Click(Sender: TObject);
begin
  with Sender as TSpeedButton do
    Label1.Caption := Caption;
end;
```

```
end.
```

اتبع هذه الخطوات لإعادة إنشاء برنامج palette وتكوين الـ tabled toolbar باستخدام الـ Notebook:

١- أضف الـ Notebook على الـ form، وحدد خاصية الـ Align به - alTop. قم بتغيير الـ Height الخاص بالـ Notebook إلى ٢٥.

٢- أضف TabSet على الـ form. حدد خاصية الـ Align له بـ alTop. هذا يضع الـ TabSet تحت الـ Notebook مباشرة.

٣- اختر خاصية الـ Tabset's Tabs قم بتغيير الـ Tabs للـ TabSet1، واضغط الزر البيضاوي لتشغيل الـ String list editor الخاص بـ Delphi. ادخل الـ toolbar labels، واحدة لكل سطر (لإعادة إنشاء البرنامج، أدخل First، Second، Third، Fourth، Fifth).

٤- اختر خاصية الـ Pages للـ Notebook1، واضغط الزر البيضاوي لفتح الـ Notebook editor الخاص بـ Delphi. اضغط زر الـ Edit لتعديل اسم الصفحة الافتراضي ليصبح First. ثم، اضغط زر الـ Add لإضافة صفحات الـ tab labels المتبقية. عند ربط الـ Notebook بـ TabSet باستخدام التقنية الموضحة هنا، بطاقة لكل Notebook page يجب ان يكون TabSet مناسبة.

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

٥- ان خاصية الـ ActivePage للـ Notebook تساوى اسم الصفحات التي ادخلتها في الخطوة السابقة . قم بتغيير الـ ActivePage لـ First . ادخل الـ SpeedButton objects في الـ Notebook1 (اضغط الفأرة داخل الـ Notebook1 ، ثم اسحب كل SpeedButton لموضعه النهائي) .

٦- اعد الخطوة رقم ٥ لصفحات Notebook اخرى . قم بتغيير الـ ActivePage لـ Second ، Third ، Fifth . . . ، وادخل بعض الـ SpeedButton في كل صفحة . يمكنك وضع label على الـ SpeedButton الخاص بك كما فعلت في المثال ، أو قم بتعيين glyph bitmaps لهم . ادخل Label و اختر حجم font كبير لإظهار الـ caption الخاص بالزر المختار .

٧- عندما تنتهى من اعداد كل صفحة Notebook ، حدد الـ ActivePage بـ First وخاصية الـ PageIndex للـ TabSet بصفر . قم بإنشاء الـ OnClick لكل الـ SpeedButton objects . ايضاً قم بإنشاء الـ OnClick للـ TabSet1 . ادخل العبارات الموجودة في القائمة . لتقلب الـ Page على الـ toolbar يقوم الـ TabSet1 handler بتعيين الـ label الـ tabled للـ ActivePage الخاصة بالـ Notebook1 . يقوم الـ SpeedButton handler بعرض الـ caption الخاص بالزر المختار .

**Tip** فكرة: لإنشاء الـ tabled toolbar في اسفل نافذة ، اعكس الخطوات رقم ١

و ٢ وحدد الـ Align لكلا الـ objects بـ alBottom .



### الـ TabbedNotebook :

ان الاختلاف الاساسى بين الـ TabbedNotebook وزوج من الـ Notebook والـ TabSet هو المظهر . استخدم الـ TabbedNotebook لإنشاء dialogs متعددة الصفحات تبدو مثل file folders ذات tabs اختيار على طول القمة . استخدم الـ Notebook والـ TabSet عندما تحتاج مزيداً من التحكم على موضع الـ object ، وعندما تحتاج مرونة برمجة عملية الانتقال بين الـ Page باستخدام components منفصلة .

يعتبر الـ TabbedNotebook مفيداً بشكل خاص في إنشاء dialogs اختيارات - على سبيل المثال ، مع طباعة الخيارات على صفحة واحدة ، Color

Setup على أخرى، وخيارات أخرى على صفحة أخرى. بتقسيم ال controls الى فئات، يمكنك برمجة امر ال Options الذى يفتح نافذة واحدة، والتي قد يجدها المستخدم اقل تداخلاً من الأوامر المتعددة فى قائمة Options.

لاستخدام ال TabbedNotebook أضفها على ال form. عادة ما تريد ان تحدد خاصية ال Align لل object بـ alClient حتى يملأ ال TabbedNotebook تماماً النافذة. ولكن هذا يرجع إليك قطعاً. اختر خاصية ال Pages لإنشاء كل صفحة ذات labeled باستخدام نفس ال Notebook editor المذكور فى الفصل السابق. على كل صفحة، أدخل ال controls التى تحتاجها (CheckBoxes، RadioButtons، و objects أخرى).

فى وقت التشغيل، يمكنك عرض صفحة معينة بطريقتين، غالباً لن تحتاج ان تفعل هذا الآن فالمستخدمون يمكن ان يختاروا الصفحات التى يريدونها. ولكن، لعرض صفحة بالرقم، قم بتعيين قيمة لل PageIndex. ان الفهرس الأول هو صفر، لذا فيمثل رقم اثنين الصفحة الثالثة:

`TabbedNotebook1.PageIndex := 2; {Display page 3}`

أو، لعرض صفحة باسم ال labeled، قم بتعيين ال string لل  
: ActivePag

`TabbedNotebook1.ActivePage := 'General options';`

للحصول على نافذة أكثر ترتيباً فى dialog الخيارات المعقد ذى الصفحات الكثيرة، قم بتعيين قيمة لل TabsPerRow. على سبيل المثال، اذا كان ال dialog الخاص بك له ١٢ صفحة، حدد ال TabsPerRow بـ ٤ لتجميع ال folders فى ثلاث صفوف من اربع. ولكن هذا يبدو انه يعمل فقط فى Delphi Versions 1 and 2. تعرض Versions 3 and 4 صفاً واحداً فقط من الابواب ال tabs (انظر ال PageControl فى هذا الباب لمعرفة طريقة افضل لإنشاء dialog متعددة الصفحات). وهذا لان Delphi 3 قد ترك ال code ال ١٦ بت القديمة لل TTabbedNotebook، واستبدلها بـ Tabs control ذى ال ٣٢ بت من ال Windows. وهذا التغيير قد أدى الى تحديد مظهر ال control، وكذلك قلل حجم ملف ال code. ولكن كنتيجة لذلك، لم يعد ال component يستطيع تحديد عدد

## الباب الثاني عشر : التعامل من خلال Dialog Boxes

ال tabs لكل صف ولم يعد ممكناً تشغيل وإبطال tabs منفردة . مازال ال control قادراً على صفوف tabs متعددة- لا يمكنك فقط تحديد كم تريد . وهذه الجوانب السلبية تعتبر ضئيلة ، والطفرة في الأداء وقدرة الذاكرة على الحفظ أكبر بكثير من أي عيوب تنتج عن هذه . التغييرات .

**فكرة:** يقوم Delphi بصورة تلقائية بتعديل ال TabsPerRow بحيث تناسب كل ال labels مع ال tabs . لهذا السبب ، يجب ان تعين قيمة جديدة لإنشاء كل صفحات ال TabbedNotebook .

Tip



لتغيير font النظام الافتراضي لـ tab labels ، اختر font مختلفاً لخاصية ال TabFont . تؤثر خاصية ال Font للـ control objects المضافه لـ TabbedNotebook . قد يكون ال fonts مختلفين . على سبيل المثال ، يمكنك استخدام نص مائل لـ tab labels ونص عادي للـ control على ال tabbed Pages .

قد تنتمي ال component objects الى form حتى عندما تبدو كأنها مستقرة داخل ال TabbedNotebook . على سبيل المثال ، أضف زر Close على ال form واسحبه فوق ال TabbedNotebook . يظهر الزر في أعلى صفحة الكتيب ، ولكنه مازال ينتمي للـ form . وبسبب علاقته بالـ Parent object ، يبقى الزر مرئياً في كل صفحات الكتيب . وهذا يعمل حتى عندما يملأ ال TabbedNotebook تماماً ال client area بالـ form .

بعد تحديد خاصية ال Align للـ TabbedNotebook بـ alClient ، من الصعب ان تضيف objects على ال form ، ولكن هناك حل سهل . حدد مؤقتاً ال Align بـ alNone . اختر ال Button أو ال BitBtn (أو أي component آخر تريد إضافته) في ال VCL palette ثم قم بتكبير نافذة ال form . اضغط مؤشر الفأرة في خلفية ال form لإضافه ال object الخاص بك ، واسحب الزر الناتج على أي صفحة TabbedNotebook (لا يهم أي واحدة) . قم بتقليص النافذة مرة أخرى لحجمها الطبيعي ، وحدد خاصية ال Align للـ TabbedNotebook مرة أخرى بـ alClient . عندما تقوم بتشغيل البرنامج ، يبقى الزر مرئياً بغض النظر عن أي صفحة من الصفحات تم اختيارها .

أو، استمر في ضغط مفتاح Shift عندما تضغط ال form لاسقاط ال component . ان ضغط ال Shift يسقط ال object فى ال Parent object للحاوية المستهدفة (ال form فى الغالب) بدلاً من الحاوية نفسها .

ان احدى الاستخدامات العامة لل TabbedNotebook هى حمل ال Memo ، واحداً فى كل صفحة . إدخال ال Memos فى صفحات ال TabbedNotebook ، وحدد خصائص ال Align لل Memo بـ alClient . ولأن ال TabbedNotebook ، يملك ال Memos ، فهى تملأ الصفحة ولكن لا تؤثر على ال tabbed labels . وهذه طريقة سريعة لإنشاء text editor متعدد الصفحات ، ولكن عليك ان تقوم ببرمجة ال method بحذر لأداء عمليات على الصفحة الحالية فقط . على سبيل المثال ، لنسخ نص الى ال clipboard ، افحص ال PageIndex لتحديد أى Memo تستخدم :

```
case TabbedNotebook1.PageIndex of
  0: Memo1.CopyToClipboard;
  1: Memo2.CopyToClipboard;
  2: Memo3.CopyToClipboard;
end;
```

هذا الحل يعمل ولكنه ليس الأمثل . توضح القائمة (١٢-٦) طريقة أكثر عمومية تعمل بغض النظر عن عدد الصفحات التى يملكها ال TabbedNotebook . يقوم البرنامج بتعيين المتغير P ، من نوع ال TComponent ، نتيجة ال FindComponent ، التى تبحث عن ال Memo باستخدام ال PageIndex لإنشاء اسماء ال object (Memo1 ، Memo2 ، . . . ، MemoN) . فإذا نجح البحث ، يقوم البرنامج باستدعاء ال CopyToClipboard لل Memo الموجودة على الصفحة الحالية . قد تستخدم code شبيهة فى أمر قائمة ال EditlCopy أو OnClick للزر .

**القائمة (١٢-٦): لنسخ نص Memo مختار الى ال Clipboard ، استخدم ال FindComponent لوضع ال Memo على صفحة ال TabbedNotebook الحالية ، ثم استدع ال CopyToClipboard لل Memo**  

```
procedure TForm1.Copytoclipboard1Click(Sender: TObject);
var
```

## الباب الثاني عشر : التعامل من خلال ال Dialog Boxes

```
P: TComponent;  
begin  
  P := FindComponent('Memo' +  
    IntToStr(TabbedNotebook1.PageIndex + 1));  
  if P <> nil then  
    TMemo(P).CopyToClipboard;  
end;
```

**فكرة:** ان اختيار component بقائمة اللائحة الخاصة بال Object Inspector ينتقل بصورة تلقائية الى صفحة ال Notebook أو ال TabbedNotebook التي تحتوى هذا ال object .



### إدخال صفحات اثناء وقت التشغيل:

**على القرص المدمج:** ان تطبيق ال AddPage على القرص المدمج في دليل ال Source\AddPage يوضح كيفية إضافة صفحات جديدة ال TabbedNotebook في وقت التشغيل . يوضح البرنامج ايضاً كيفية إدخال control في صفحة - مرة أخرى ، تحت تحكم البرنامج تماماً . تظهر القائمة (١٢-٧) ال source code البرنامج . قم بتشغيل ال AddPage واضغط ال AddPage لإدخال صفحات جديدة في ال TabbedNotebook . اضغط ال Add control لإدخال ListBox في الصفحة الحالية .



### القائمة (١٢-٧): AddPage\main.Pas

```
unit Main;  
  
interface  
  
uses  
  SysUtils, Windows, Messages, Classes, Graphics,  
  Controls, Forms, Dialogs, Buttons, StdCtrls, TabNotBk;  
  
type  
  TMainForm = class(TForm)
```



```

TabbedNotebook1: TTabbedNotebook;
  AddPageButton: TButton;
  CloseBitBtn: TBitBtn;
  AddControlButton: TButton;
  procedure AddPageButtonClick(Sender: TObject);
  procedure AddControlButtonClick(Sender: TObject);
private
  { Private declarations }
  public
  { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

{ Insert new page into TabbedNotebook }
procedure TMainForm.AddPageButtonClick(Sender: TObject);
begin
  with TabbedNotebook1 do
    Pages.Add(Format('Page %d',[Pages.Count-1]));
end;

{ Insert new control into current page }
procedure TMainForm.AddControlButtonClick(Sender:
TObject);
var
  L: TListBox;
  P: TWinControl;
begin
  L := TListBox.Create(Self);
  with TabbedNotebook1 do
    begin

```

## الباب الثاني عشر : التعامل من خلال Dialog Boxes

```
P := Pages.Objects[PageIndex] as TWinControl;  
L.Parent := P;  
L.SetBounds(10, 10, 100, 100);  
{ Insert page tab label into edit control for demonstration.  
  You don't have to perform this step. }  
L.Items.Add(TTabPage(P).Caption);  
end;  
end;  
  
end.
```

يوضح الـ `AddPageButtonClick Procedure` كيفية إضافة صفحة للـ `TabbedNotebook`. ببساطة، أضف String الخاصية الـ `Pages`. يستدعى البرنامج الـ `Format` لإنشاء string labels مثل `Page 1`، `Page 2`، ...، `Page n`.

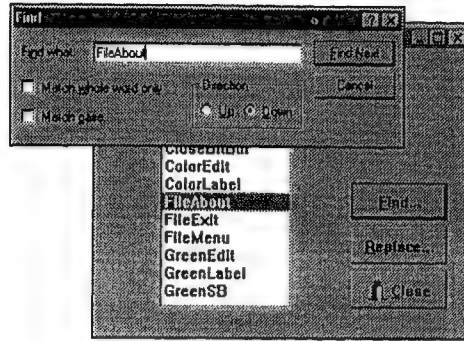
يوضح الـ `AddPageButtonClick Procedure` كيفية إضافة control object جديد لصفحة الـ `TabbedNotebook` الحالية. أولاً، ينشئ الـ `Create` الـ `constructs` الجديد. ثم بعد ذلك، يتم تحديد الـ `P` لتشير الى الصفحة، باستخدام الـ `Objects array` في قائمة string التابعة للـ `TabbedNotebook Pages`. يعين البرنامج الـ `Page reference (P)` كـ `control's parent`، ثم يستدعى الـ `SetBounds` لتحديد مكان، وارتفاع، وعرض الـ `control`. (والأنواع الأخرى من الـ `control objects` تتطلب أنواع بدايات أخرى). أخيراً، لتعرض شيئاً في الـ `control` الجديد، يضاف الـ `Add method` لـ `tab label` الصفحة الحالية للـ `ListBox` الذي تم إدخاله حديثاً. لا يجب عليك ان تؤدي هذه الخطوة الأخيرة.

### من خلال الـ Dialogs:

ان الـ `ReplaceDialog` والـ `FindDialog` التابعة لـ `Delphi` سهلة للغاية. الجزء الصعب نوعاً ما هو كتابة code البحث والابدال التي تعمل بالارتباط بـ `modeless dialog boxes`. بعد هذه المقدمات القصيرة، يقدم تطبيق غلافاً يمكنك استخدامه لتنفيذ اوامر الـ `Replace` والـ `Find`.

## الـ Find dialog:

يقدم الـ FindDialog component أثنان من methods و event. واحد. لعرض نافذة الـ display الموضحة فى الشكل (١٢-٤) وبدء البحث، استدع الـ Execute method: استدع الـ CloseDialog لإخفاء الـ display. لأداء البحث الفعلى، اكتب للـ OnFind، الذى تم توليده عندما ضغط المستخدمون زر الـ Find Next.



شكل (١٢-٤): الـ FindDialog component يعرض modeless dialog.

للحصول على النص الذى ادخله المستخدمون فى الـ Find لأى edit control، استخدم خاصية الـ Find Next للـ FindDialog. يمكنك كذلك اختيار Options متنوعة- على سبيل المثال، لإخفاء وإبطال خيار الـ match case check Box.

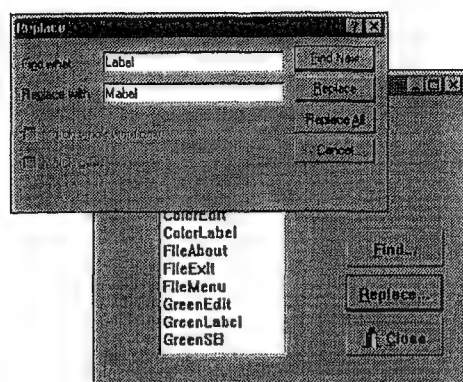
## find and replace dialog:

كما هو موضح فى شكل (١٢-٥)، يعد الـ ReplaceDialog نسخة ممتدة للـ FindDialog، بالإضافة إلى edit control ثانى، لإدخال نص الإبدال. لأعضاء الـ FindDialog، يضيف الـ ReplaceDialog خاصية أخرى، وهى ReplaceText، والتى تحتوى على النص الذى ادخله المستخدمون فى الـ Replace بنافذة الـ edit. استدع الـ Execute لعرض الـ ReplaceDialog. استدع الـ CloseDialog لإخفاء نافذة الـ display.

يتعرف الـ ReplaceDialog على events. إنشئ OnFind event handler للبحث عن بنود النص. إنشئ OnReplace event handler لأداء

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

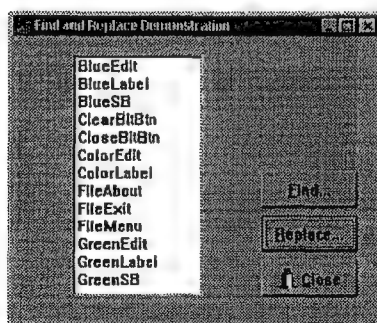
الإبدال ان كتابة هذا الـ code يتطلب ايضاً بعض الحيل ، كما سأوضح فى الفصل التالى .



شكل (١٢-٥) : الـ ReplaceDialog component يعرض . modeless dialog

### برمجة اوامر Find والـ Replace :

على القرص المدمج؛ لسوء الحظ ان برمجة اوامر الـ Find والـ Replace اصعب بكثير من مجرد إضافة إثنين من الـ components على form . لتبسيط الأمر ، اتبع الخطة الموضحة فى تطبيق الـ FindRepl على القرص المدمج . يوضح شكل (١٢-٦) البرنامج . إنه يذكر بعض اسماء الـ object (تم قصها من الـ SysColor Utility) فى الـ ListBox . اضغط زر الـ Find لتجد العناصر فى الـ ListBox اضغط Replace لتستبدل البنود أو جزء منها . توضح القائمة (١٢-٨) كيف يقوم البرنامج باداء هذه العمليات .



شكل (١٢-٦) : يوضح تطبيق الـ FindRepl البرمجة اللازمة لأوامر الـ Find والـ Replace التابعة للتطبيق

القائمة (١٢-٨): FindReplMain.pas

```
unit Main;

interface

uses
    SysUtils, Windows, Messages, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls, Buttons;

type
    TMainForm = class(TForm)
        ListBox1: TListBox;
        FindDialog: TFindDialog;
        FindBitBtn: TBitBtn;
        CloseBitBtn: TBitBtn;
        ReplaceBitBtn: TBitBtn;
        ReplaceDialog: TReplaceDialog;
        procedure FindBitBtnClick(Sender: TObject);
        procedure FindDialogFind(Sender: TObject);
        procedure ReplaceBitBtnClick(Sender: TObject);
        procedure ReplaceDialogFind(Sender: TObject);
        procedure ReplaceDialogReplace(Sender: TObject);
    private
        FindIndex, FoundPos, FoundLen: Integer;
        FoundItem: Boolean;
    public
    end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

{ Begin a FindDialog operation }
```

## Dialog Boxes الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

```
procedure TMainForm.FindBitBtnClick(Sender: TObject);
begin
    FindDialog.Execute;
    FindIndex := 0;
    ListBox1.ItemIndex := -1;
end;

{ Continue a FindDialog operation }
procedure TMainForm.FindDialogFind(Sender: TObject);
var
    S: String;
begin
    while FindIndex < ListBox1.Items.Count do
    begin
        S := ListBox1.Items[FindIndex];
        Inc(FindIndex);
        if Pos(FindDialog.FindText, S) <> 0 then
        begin
            ListBox1.ItemIndex := FindIndex - 1;
            Exit;
        end;
    end;
    ShowMessage('No more matches!');
    FindDialog.CloseDialog;
end;

{ Start a ReplaceDialog operation }
procedure TMainForm.ReplaceBitBtnClick(Sender: TObject);
begin
    ReplaceDialog.Execute;
    FindIndex := 0;
    ListBox1.ItemIndex := - 1;
    FoundItem := False;
end;

{ Continue a ReplaceDialog operation }
```

```

=====
procedure TMainForm.ReplaceDialogFind(Sender: TObject);
var
  S: String;
begin
  while FindIndex < ListBox1.Items.Count do
    begin
      S := ListBox1.Items[FindIndex];
      Inc(FindIndex);
      FoundPos := Pos(ReplaceDialog.FindText, S);
      if FoundPos <> 0 then
        begin
          ListBox1.ItemIndex := FindIndex - 1;
          FoundLen := Length(ReplaceDialog.FindText);
          FoundItem := True;
          Exit;
        end;
      end;
    ShowMessage('No more matches!');
    ReplaceDialog.CloseDialog;
  end;

```

```

{ Perform replacement for a ReplaceDialog operation }
procedure TMainForm.ReplaceDialogReplace(Sender: TObject);
var
  S: String;
begin
  if frReplaceAll in ReplaceDialog.Options then
    ShowMessage('Replace All not implemented')
  else if not FoundItem then
    ShowMessage('Click Find to begin/continue search')
  else begin
    S := ListBox1.Items[FindIndex - 1];
    Delete(S, FoundPos, FoundLen);
    Insert(ReplaceDialog.ReplaceText, S, FoundPos);
    ListBox1.Items[FindIndex - 1] := S;
    FoundItem := False;
  end;

```

## الباب الثاني عشر : التعامل من خلال Dialog Boxes

```
end;
```

```
end;
```

```
end.
```

ان برمجة ال Find هي اسهل المهتمين . أولاً ادخل event handler لأمر أو زر يبدأ البحث . استدع ال Execute method لل FindDialog وقم باعداد متغيرات عمومية تستخدم لاستكمال البحث . على سبيل المثال ، يبدأ ال FindRepl البحث بهذه العبارات :

```
FindDialog.Execute;
```

```
FindIndex := 0;
```

```
Listbox1.ItemIndex := -1;
```

يقوم السطر الاول باحضار modeless dialog . ويقوم السطر الثانى بإنشاء متغير عام يستخدم للإشارة لسطر ال Listbox الذى تم بحثه مؤخراً . والسطر الأخير يحدد ال ItemIndex فى ال Listbox بـ (-1) بحيث لا يكون هناك اسطر يتم إبرازها من البداية .

عندما تبدأ البحث بضغط زر ال Find Next لل dialog ، يقوم ال FindDialogFind بأداء البحث الفعلى باستخدام ال FindIndex العام لتحديد أى من عناصر ال Listbox يختبر . تقوم ال Pos باختبار ما اذا كانت خاصية ال FindText الموجودة فى ال string المستهدف S ، منسوخة من ال Items الخاصة بال Listbox . اذا كانت ال Pos غير صفرية ، يوجد تجانس ، والتعيين لل ItemIndex يبرز العنصر المتجانس .

لاحظ ان event handler يوجد بعد إيجاد التجانس . هذا لا يغلق ال dialog - إنه فقط يعيد ال control الى البرنامج بحيث يمكن للمستخدمين من اداء اعمال اخرى . ولكن ، اذا كان ال FindIndex العام يساوى عدد العناصر ، يستدعى البرنامج ال ShowMessage ليخبر المستخدمين ان البحث قد انتهى . يقوم ال CloseDialog عندئذ باخفاء نافذة ال FindDialog .

ان كتابة code لأمر الابدال يعتبر اصعب . يبدأ البحث كما مع ال Find ، ولكن ابدأ أى متغيرات عامة تحتاجها لعمل الابدال . (انظر ال



ReplaceBitBtnClick فى القائمة). انك تحتاج الى Boolean Flag ، مثل ال FoundItem ، للإشارة الى ان بنداً قد تم إيجاداه. إبدأ هذا المتغير بـ False فى بداية البحث. وهنا يظهر كيف يستجيب البرنامج لضغط زر ال Replace :

ReplaceDialog.Execute;

FindIndex := 0;

Listbox1.ItemIndex := -1;

FoundItem := False;

يقوم السطر الأول باستدعاء ال Execute لعرض . modeless dialog box . يعين السطر الثانى صفراً للـ FindIndex والذى يمثل سطر ال Listbox الذى يتم بحثه . يعين السطر الثالث (-1) لخاصية ال ItemIndex للـ Listbox بحيث لا يتم ابراز اية اسطر . يبدأ السطر الأخير بال Boolean flag (FoundItem) ، بـ False .

لإنهاء امر ال Replace ، إنك تحتاج الى اثنين من event handler . الأول (وهو ReplaceDialogFind فى القائمة) يشبه للـ FindDialog event handler . عند إيجاد عنصر متجانس ، ابدأ المتغيرات العامة التى تحتاجها لأداء إبدال . حدد FoundItem flag ، بـ True بحيث يمكن ال event handler الثانى ان يحدد أنه تم استدعاؤه كنتيجة للبحث المتجانس .

فى هذا event handler (انظر ReplaceDialogFind) ، اذا كان ال FoundItem ، بـ True ، ينفذ ال procedure عبارات الإبدال الخاصة به . كما توضح القائمة ، يمكنك ايضاً فحص ما اذا كان المستخدمون قد ضغطوا زر ال Replace All باختيار ما اذا كان ثابت ال frReplaceAll موجود فى مجموعة ال Options للـ dialog . بالطبع ، يعتبر code الابدال الفعلى فريداً لبرنامجك ، ولكن Delphi لا يستطيع ان يفعل كل شئ (أليس كذلك؟) .

## : Page Controls

ان ثلاثة components جديدة نسبياً قد يسرت مهمة إنشاء dialogs ونوافذ اخرى متعددة الصفحات وهذه ال components الثلاثة هي :

• **PageControl** : عندما تحتاج الى إنشاء نافذة أو dialogs متعدد الصفحات بمجموعات منفصلة من ال controls فى كل صفحة ، فإن هذا ال component هو افضل اختيار .

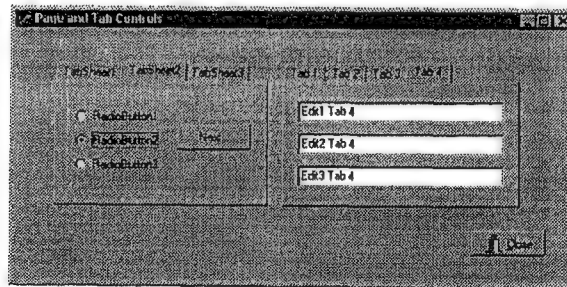
## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

• **TabSheet** : تعتبر كل صفحة في نافذة أو الـ PageControl dialog هو عبارة عن TabSheet . هذا الـ component غير موجود على الـ VCL palette - انك تنشئ الـ TabSheet objects بالضغط على الزر الأيمن للفأرة داخل الـ PageControl واختيار امر الـ New Page من القائمة الناتجة .

• **TabControl** : عندما تحتاج dialog أكثر بساطة ، single-component ، متعدد الصفحات ، استخدم هذا الـ component بدلاً من الـ PageControl و الـ TabSheet . ان النتائج النهائية تعتبر متشابهة بصرياً ، ولكن وضع الـ component على الصفحات المنفردة يعتبر مسئوليتك . تعتبر الـ TabControls مفيدة في إنشاء نوافذ متعددة الصفحات بنفس الـ controls في كل صفحة ، ولكن بمحتويات مختلفة - مجموعة من الـ Edit ، مثلاً ، التي تعتبر محتوى نصها اعتماداً على أى صفحة يختار المستخدم .

توضح الفصول التالية كيفية استخدام كلاً من الـ components الثلاثة . ان الـ PageControl والـ TabControl موجودان على الـ Win32 palette يتم إنشاء الـ TabSheet باستخدام الـ PageControl كما ستوضح في الفصل التالي .

**على القرص المدمج** : يوضح شكل (١٢-٧) أمثلة من الـ PageControl والـ TabControl . ان الصفحات الموجودة في الـ PageControl تعتبر الـ TabSheet . هذا الـ objects يوضح مظهر الـ PageControl ، الموجود في دليل الـ Source\PageTab على القرص المدمج .  
توضح القائمة (١٢-٩) الـ source code لـ PageTab .



شكل (١٢-٧) : برنامج العرض PageTab يوضح نماذج من الـ PageControl (يساراً) والـ TabControl (يميناً) . كل صفحة من الـ PageControl تعتبر TabSheet

القائمة (١٢-٩) PageTabMain.pas

unit Main;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, StdCtrls, ComCtrls, Buttons;

type

```
TMainForm = class(TForm)
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    CheckBox3: TCheckBox;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    TabControl1: TTabControl;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    BitBtn1: TBitBtn;
    TabSheet3: TTabSheet;
    DateTimePicker1: TDateTimePicker;
    Button2: TButton;
    RadioButton3: TRadioButton;
    Button1: TButton;
    procedure TabControl1Change(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
```

```
end;  
  
var  
    MainForm: TMainForm;  
  
implementation  
  
{$R *.DFM}  
  
procedure TMainForm.TabControl1Change(Sender: TObject);  
var  
    S: String;  
begin  
    S := IntToStr.TabControl1.TabIndex + 1);  
    Edit1.Text := 'Edit1 Tab ' + S;  
    Edit2.Text := 'Edit2 Tab ' + S;  
    Edit3.Text := 'Edit3 Tab ' + S;  
end;  
  
procedure TMainForm.Button1Click(Sender: TObject);  
begin  
    PageControl1.ActivePage := TabSheet3;  
end;  
  
procedure TMainForm.Button2Click(Sender: TObject);  
begin  
    PageControl1.ActivePage := TabSheet1;  
end;  
  
end.
```

### ال PageControl component :

يسيطر ال PageControl عملية إنشاء dialogs متعددة الصفحات ذات مجموعات فريدة من ال controls على كل صفحة . لتري مثالاً على ال PageControl اثناء عمله ، قم بتحميل ال PageTab.dpr فى Delphi من دليل

ال Source\PageTab . يعرض البرنامج ال PageControl (يساراً) وال TabControl (يميناً) . قبل تشغيل البرنامج لاحظ انه بإمكانك اختيار ال PageControl tabs فى وقت التصميم ، ولكن لا يمكنك ان تفعل هذا مع ال TabControl . . وهذا لان صفحات ال PageControl تعتبر TabSheet ، ولذلك ، يمكن اختيارها وتعديلها باستخدام ال Object Inspector . ولكن ال TabControl يعتبر object واحد- ان صفحاته ما هى الا خداع- ولذلك ، لا يمكنك اختيار ال tabs المنفردة فى وقت التصميم .

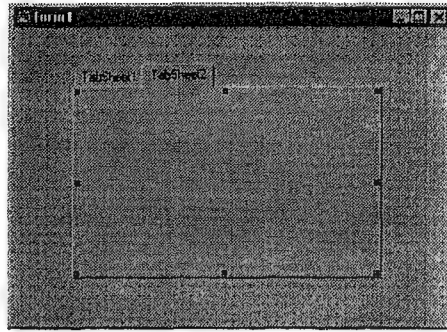
حاول تجربة الخطوات التالية لتعرف المزيد عن استخدام ال PageControl :

١- ابدأ تطبيقاً جديداً . اختر ال Win32 page tab على ال VCL palette لـ Delphi واضغط ال PageControl (الثانى من اليسار) . اضغط داخل ال from ال PageControl . يبدو ال object فى البداية شبه اللوحة الخالية .

٢- لإنشاء صفحات فى ال PageControl ، اضغط زر الفأرة الأيمن داخل ال PageControl . اختر New Page من القائمة الناتجة . هذا يؤدى الى إنشاء TabSheet ، أضف ذلك ال object الى أى objects اخرى مملوكة للـ PageControl هذا ، وعرض two pages . يوضح شكل (١٢-٨) العرض بعد إضافة صفحتين .

٣- اضغط ال page tabs ولاحظ ان نافذة ال Object Inspector تعرض خصائص للـ PageControl1 . اضغط داخل نافذة ال PageControl (تحت ال tabs) لاختيار ال TabSheet للصفحة . يظهر ال Object Inspector عندئذ خصائص ال TabSheet .

٤- لإضافة control للصفحة ال PageControl ، اختر الصفحة ثم إسقط أى control عليها . يمكنك ان تفعل هذا مع أى control تقريباً ، مثل ال Buttons ، CheckBoxes ، Labels ، ال StringGrids ، وال DateTimePickers . بالرغم من ان ال controls تستقر على صفحات ال TabSheet المنفردة ، فإنك تتوصل إليها بنفس طريقة وصولك الى أى controls بالإشارة الى اسمائها مثل Button1 و CheckBox3 . لا يجب عليك ان تستخدم ال object PageControl للوصول الى ال controls التى يحتوئها .



شكل (١٢-٨): ال PageControl component مع اثنين من ال TabSheet

### خصائص ال PageControl:

يوفر ال PageControl خصائص يمكنك استخدامها للتعامل مع ال controls بطرق متنوعة . باستخدام ال PageControl من الفصل السابق ويمكنك اسقاط واحداً على ال from الخالية) ، قم بتجربة مواصفات الخصائص التالية :

● **ActivePage** : حدد هذه باسم ال TabSheet الذي تريد عرضه على انه الصفحة النشطة الأولى . في وقت التشغيل ، يمكن لعبارة program ايضاً ان تعين هذه القيمة لتغيير الصفحة النشطة تحت تحكم البرنامج [انظر ال Button2Click procedure في القائمة (١٢-٩) كمثال] .

● **DockSite** : حدة بـ True لإنشاء PageControl يعمل كـ docking station ، لل controls الاخرى . انظر "إنشاء ال Docking Controls" في هذا الباب لمزيد من المعلومات حول إنشاء docking sites ، وهي ميزة جديدة في Delphi 4 .

● **Hint** : أدخل نصاً للعرض . عندما يبقى المستخدم مؤشر الفأرة على ال controls للحظة أو اثنين . استخدم هذه الخاصية مع ال ShowHint لعرض نفس نص الملاحظة بغض النظر عن الصفحة النشطة . انظر خصائص ال Hint وال ShowHint لل TabSheet لمزيد من المعلومات .

● **HotTrack** : حددها بـ True لتوفير تغذية خلفية بصرية . عندما يمر المستخدم مؤشر الفأرة على ال page tab ، يتم إختفاء نصها لوقت قصير ليرى المستخدمون ان بإمكانهم ضغط الفأرة لاختيار ذلك ال tab .

● **MultiLine**: فى ال PageControls المعقدة ذات الصفحات الكثيرة، حدد هذه الخاصية بـ True لعرض الصفحات على صفين أو أكثر اذا لم تكن جميعها تتناسب على صف واحد بداخل عرض النافذة المحددة. عندما تكون هذه الخاصية محددة بـ False، تظهر ال Scroll arrows اذا كان هناك صفحات كثيرة جداً بحيث يصعب ان تتناسب على صف واحد- ولكن، الصفوف المتعددة تعتبر افضل فى معظم الحالات.

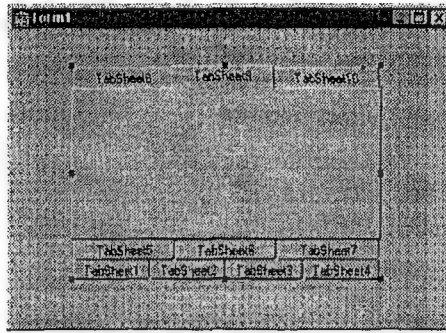
● **MultiSelect**: هذه خاصية غامضة تعمل فقط عندما يكون نمط ال tab محدد بـ Button أو tsFlatButton. إنها تبدو كسمة من سمات ال TabControl يتم عرضها فى ال Page control، ولكن اختيار صفحات متعددة لا يبدو كعملية ذات قيمة عملية كبيرة. إنها غير مؤقتة، ولا يبدو ان لها تأثيراً كبيراً، إن وجد أصلاً. مسمى هذه الخاصية باللغز.

● **OwnerDraw**: عندما يتم تحديد هذه الخاصية بـ True، فإنها مسؤوليتك أن تعرض نصاً أو رسوماً جرافيكية (أو كلاهما) فى كل tab. لتفعل هذا، قم بإنشاء OnDrawTab event handler وأدخل code لعرض نص أو bitmaps فى مساحة page tab المخططة بواسطة Rect parameter. أنظر «إنشاء ال owner draw PageControls» فى هذا الباب للحصول على مزيد من المعلومات.

● **ScrollOpposite**: عندما يكون لل PageControl tabs عديدة، يمكنك تحديد ال ScrollOpposite بـ True أو False لتغيير الطريقة التى تتحرك بها tabs أخرى، بعيداً عن الطريق عندما يختار المستخدمون tab معيناً. على سبيل المثال، فى PageControl ذى صفين من ثلاث tabs، اذا كان ال ScrollOpposite محدد بـ True، فإن اختيار tab فى إحدى الصفوف يحرك الصف الآخر الى أسفل العرض [أنظر شكل (١٢-٩)].

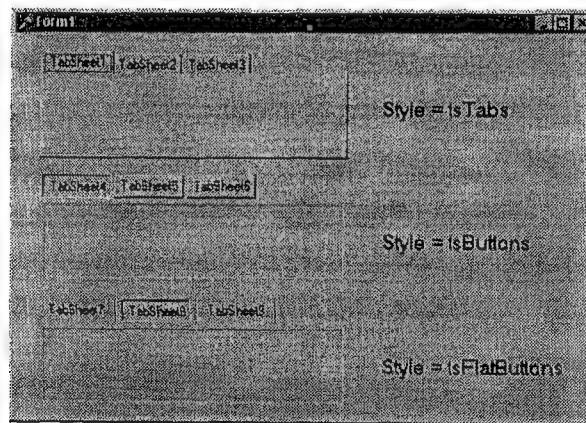
● **ShowHint**: حدد هذه الخاصية بـ True لعرض النص الموجود فى خاصية ال Hint عندما يبقى المستخدم مؤشر الفأرة على ال PageControl للحظة أو اثنتين. لعرض نص مختلف اعتماداً على أى صفحة تكون نشطة، حدد ال ShowHint بـ False واستخدم خصائص ال Hint وال ShowHint فى ال TabSheet (أنظر الفصلين التاليين).

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes



شكل (٩-١٢): تحديد الـ ScrollOpposite للـ PageControl بـ True يؤدي الى Scroll tabs المتعددة الى النهاية المعاكسة للـ control عندما يختار المستخدمون tab في مجموعة أخرى

● **Style:** هذه الخاصية تؤثر على طريقة عرض الـ tabs [شكل (١٠-١٢)]  
يوضح نماذج لكل صفة) حدد الـ tsTabs للـ control الذي يبدو مثل page tabs على folders الملفات؛ استخدم الـ tsButtons لعرض page tabs على ازرار الـ Windows؛ استخدم الـ tsFlatButtons لعرض page tabs دون ايه حدود، ولكن عند اختيارها. (فكرة: عند استخدام غط الـ tsFlatButtons، يجب ان تحدد الـ Hottrack بـ True لعرض تغذية خلفية في الـ form للأزرار المتبقية عندما يمرر المستخدم مؤشر الفأرة فوق الـ control page tabs).



شكل (١٠-١٢): استخدام خاصية الـ PageControl Style لتؤثر على طريقة عرض الـ tabs



● **TabPosition**: يمكنك استخدام هذه الخاصية لاختيار ما اذا كنت ستعرض page tabs فى أعلى الـ control (tpTop) أو اسفله (tpBottom). يتم عرض عادة فى أعلى الـ Control وكثير من المستخدمين يتوقعون ان يجدوها فى هذا الموضع. من الأفضل الا تغير هذه الخاصية الا اذا كان لديك سبب وجيه.

**Tip** **فكرة:** عندما تكون خاصية الـ Style للـ PageControl محدد بـ tsButtons أو tsFlatButtons، لا يظهر أى border حول الصفحات المنفردة للـ control. اذا كنت تريد حداً، أدخل Panel فى كل صفحة ثم قم بإسقاط الـ controls على الـ Panels. أو احفظ لـ handle الـ Window واستخدم Bevel بدلاً منه.

### الـ TabSheet component

كل صفحة على الـ PageControl تعتبر فى حد ذاتها object من الـ TTabSheet class. كما ذكرت، الطريقة الوحيدة لإنشاء TabSheets هى بالضغط على الزر الأيمن للفأرة فى الـ PageControl واختيار امر الـ New Page من القائمة. لاختيار الـ TabSheet فى الـ PageControl، اضغط الـ page tab الذى تريده، ثم اضغط داخل الـ control body (تحت الـ page tabs اذا كانوا على الـ control). يمكنك عندئذ استخدام الـ Object Inspector لعمل تغييرات لخصائص الـ TabSheets المنفردة.

**Tip** **فكرة:** عند تحديد خصائص لـ TabSheets متعددة، من الصعب، ولكن ممكن، أن تختارها جميعاً فى نفس الوقت. لتفعل ذلك، استمر فى ضغط مفتاح الـ Shift أثناء الضغط داخل كل الـ page tab، ثم فى كل الـ TabSheet (فى الـ control body). بعد اختيار الـ TabSheet النهائى، بينما مازلت تضغط الـ Shift، اضغط أى الـ page tab لإبطال اختيار الـ PageControl نفسه. يمكنك عندئذ تعيين قيم خاصة لكل الـ TabSheets المختارة حدد خاصية الـ Cursor بـ crCross مثلاً.

### خصائص الـ TabSheet:

فيما يلى خصائص الـ TabSheet مختارة والتي قد تريد تغييرها عند تصميم الـ PageControl. لتجربة هذه المواصفات، قم بإسقاط الـ PageControl على

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

form، اضغط على الزر الأيمن للفأرة داخل الـ control، واختر New Page لإضافة صفحة واحدة أو أكثر. اختر صفحة الـ TabSheet بالضغط داخل الـ PageControl (تحت الـ PageControl)، قم بتجربة مواصفات الخصائص هذه :

● **BorderWidth:** حددها بقيمة موجبة غير صفرية، لتغيير عرض الـ Border حول صفحة الـ TabSheet هذه.

● **Caption:** اكتب أى نص تريده ليظهر فى tab هذه الصفحة. يتم تعديل عرض tab بصورة تلقائية ليسع النص. إذا كان مجال الـ OwnerDraw التابع للـ PageControl محدد بـ True، فإنها مسئوليتك أن تعرض نص.

● **Cursor:** اختر واحداً من الـ cursor المذكورة فى قائمة اللائحة التابعة لهذه الخاصية. تتغير الـ cursor بصورة تلقائية عندما تمر على الصفحة.

● **Hint:** أضف الـ hint text ليظهر فى موضع الـ cursor عندما تستقر للحظة أو إثنين. حدد الـ ShowHint بـ True لعرض نص الـ Hint هذا.

● **PageIndex:** هذا الحقل محدد بصورة تلقائية بقيمة الفهرس لكل صفحة TabSheet. والفهرس للصفحة الأولى هو صفر، والتالية واحد، الى آخره. إذا قمت بتغيير هذه القيمة فى وقت التصميم، يتم تعديل الصفحات الأخرى طبقاً لذلك، ويتم إدخال الصفحة التى تغيرت فى موضع جديد. يمكنك فعل هذا بعد إنشاء الـ PageControl لتغيير نظام صفحاته المنفردة، ولكن فى أغلب الحالات لن تحتاج الى إدخال قيم فى هذه الخاصية.

**ملحوظة:** تشير الـ online help الى أن الـ PageIndex محدد بـ (-1) عندما يكون الـ Tab Visible محدد بـ False. والأمر ليس كذلك حيث تبقى قيمة الـ PageIndex كما هى بغض النظر عما إذا كانت صفحة الـ TabSheet مرئية أم لا.



● **PopupMenu:** أضف الـ PopupMenu من الـ Standard palette على الـ form، يمكن للمستخدمين عندئذ أن يضغطوا يمين الفأرة أثناء الإشارة بالـ cursor داخل صفحة الـ TabSheet لعرض قائمة. يمكن لكل الصفحات أن تستخدم نفس

ال PopupMenu، أو يمكن أن يكون لكل صفحة object مختلفاً. هذه التقنية تعتبر طريقة عظيمة لتوفير مجموعات مختلفة من الأوامر المحلية اعتماداً على الصفحة النشطة.

• **ShowHint:** حددها بـ True لعرض النص الذى تم إدخاله فى خاصية ال Hint عندما يبقى المستخدم مؤشر الفأرة على هذه الصفحة لعدة ثوان. استخدم هذه الخاصية وخاصية ال Hint لعرض hint فريدة اعتماداً على الصفحة النشطة.

**Tip** فكرة: لعرض نفس ال hint لكل صفحات ال TabSheet، من السهل أن تحدد ال ShowHint بـ False، وال ParentShowHint بـ True. اختر ال PageControl، أدخل نص ال Hint الخاص بك، وحدد خاصية ال ShowHint لهذا ال control بـ True.

• **TabVisible:** حدد هذه الخاصية بـ False فى وقت التشغيل حتى تزيل مؤقتاً ال tab sheet من الصفحة. هذا أيضاً يحدد خاصية ال TabIndex للـ Sheet بـ (-1). إن إعادة تحديد ال TabVisible بـ True مرة أخرى يظهر ال tab sheet.

**Tip** فكرة: عندما يكون ال TabVisible محدد بـ False، لم يعد من الممكن اختيار ال TabSheet باستخدام الفأرة لأن تلك الصفحة لم تعد معروفة. بدلاً من ذلك، اختر هذه ال TabSheet من قائمة اللائحة للـ Object Inspector. يمكنك عندئذ إعادة تحديد ال TabVisible بـ True لتجعل الصفحة مرئية مرة أخرى.

### إنشاء PageControls خاصة بالowner-draw

لكى تخصص عرض محتويات ال page tab، يمكنك إنشاء PageControl خاص بالowner-draw. هذا يضع مسئولية عرض النص أو الرسوم الجرافيكية للـ page tab على عاتقك - وهى ميزة قد تستخدمها، مثلاً، لعرض ال page tabs باستخدام fonts مختلفة، أو لعرض أيقونة.

**على القرص المدمج:** يوضح شكل (١٢-١١) مثلاً على ال PageControl الخاص بالowner-draw، والذى أضفت له ١٢ صفحة. يعرض البرنامج نص ال page tab النشطة بالخط المائل، ويعرض



## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

أيضاً أيقونة صغيرة بعد كل tab label . توضح القائمة (١٢-١٠) الـ source code لهذا البرنامج- وهي موجودة على القرص المدمج فى دليل الـ Source\PageTabOD . لتوفير المساحة ، قمت بحذف بضعة أسطر تقوم بمجرد تعريف الـ Label والـ TabSheet فى الـ form class- يقوم Delphi بإنشائهم ويمكنك تجاهلها .



شكل (١٢-١١) : إنه باستخدام الـ owner-draw PageControl ، يمكنك أن تعرض نص الـ page tab باستخدام font مختلف (هذا المثال يستخدم الخط المائل للصفحة النشطة) ، ورسوم جرافيك مثل الأيقونات الموضحة هنا

### القائمة (١٢-١٠) : PageTabODMain.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, ComCtrls, Images, StdCtrls;

type
  TMainForm = class(TForm)
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    // TabSheet2 ... TabSheet 12 deleted
    ImageList1: TImageList;
    Label1: TLabel;
    // Label2 ... Label12 deleted
```

=====

```

procedure PageControl1DrawTab(Control: TCustomTabControl;
  TabIndex: Integer; const Rect: TRect; Active: Boolean);
  private
  { Private declarations }
  public
  { Public declarations }
  end;

```

```

var
  MainForm: TMainForm;

```

implementation

{ \$R \*.DFM }

```

procedure TMainForm.PageControl1DrawTab(
  Control: TCustomTabControl; TabIndex: Integer;
  const Rect: TRect; Active: Boolean);
begin
  with Control.Canvas do
  begin
    if active
    then Font.Style := [fsItalic]
    else Font.Style := [];
    TextRect(Rect, Rect.Left + 20, Rect.Top + 3,
      PageControl1.Pages[TabIndex].Caption);
    ImageList1.Draw(Canvas,
      Rect.Left + 2, Rect.Top + 2, 0);
  end;
end;

end.

```

اتبع هذه الخطوات لإنشاء برنامج العرض PageTabOD المذكور هنا :  
 ١- إبدأ تطبيقاً جديداً .

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

٢- قم بإسقاط الـ ImageList من الـ Win32 على الـ form . اضغط مرتين على زر الـ component واضغط زر الـ Add لإضافة صورة أيقونة . لقد استخدمت أيقونة الـ Sample.ico من دليل الـ Source\Data على القرص المدمج ، ولكن بإمكانك أن تستخدم أى صورة أخرى تريدها .

٣- قم بإسقاط الـ PageControl من الـ Win32 على الـ form . اضغط زر الفأرة الأيمن داخل الـ component واختر New Page من القائمة . افعل هذا ١٢ مرة .

٤- إختياري : قم بإسقاط الـ Label على كل صفحة من الـ PageControl . لتفعل هذا ، اختر كل tab واضغط داخل الـ PageControl هذا يؤدي الى اختيار الـ TabSheet الذى يعمل كوعاء للـ controls هذه الصفحة (افحص نافذة الـ Object Inspector لتأكد من أنك قد اخترت الـ TabSheet) ثم ، قم بإسقاط الـ Label على الـ TabSheet . هذا يربط الـ label بالصفحة - تظهر الـ label عندما تكون الصفحة الخاصة بها نشطة فقط . استخدم نفس الطريقة لإسقاط أى أنواع controls أخرى على صفحات الـ PageControl .

٥- اختر الـ PageControl1 . يمكنك أن تفعل هذا إما باستخدام قائمة إسقاط الـ Object Inspector أو بضغط أى page tab . تأكد من أن الـ Object Inspector يذكر الـ PageControl1 فى البداية .

٦- حدد الـ Align بالـ alClient حتى يملأ الـ PageControl الـ client area بالنافذة . هذه الخطوة إختيارية ، ولكن عندما تنشئ Pages dialogs ، فإنك غالباً ما تختار هذا التحديد .

٧- حدد الـ MultiLine بـ True حتى تصبح كل page tabs مرئية فى صفوف متعددة . وكذلك حدد الـ ScrollOpposite بـ True بحيث تتحرك بعض الصفوف الى أسفل عند الحد السفلى للنافذة .

٨- حدد الـ OwnerDraw بـ True . عند هذه النقطة ، تصبح page tab labels فارغة .

٩- لعرض الأيقونات والنص فى كل page tab ، يجب أن تنشئ الـ event handler لـ PageControl . تأكد من أن الـ PageControl1 مازال مختاراً ، ثم

اضغط الـ Events tab فى الـ Object Inspector . اضغط مرتين الـ OnDrawTab لإنشاء procedure، والذي يمكنك أن تملأه بالـ code الموجودة فى القائمة .

عندما تقوم بتشغيل البرنامج التام بضغط F9، يتم استدعاء OnDrawTab لرسم النص والرسوم الجرافيكية لكل Page tab . يتم تعريف الـ procedure والـ parameters الخاصة به كما يلى :

```
procedure TMainForm.PageControl1DrawTab(
  Control: TCustomTabControl; TabIndex: Integer;
  const Rect: TRect; Active: Boolean);
```

● Control: هذا يشير الى الـ object الذى يكون صورة الـ tab . لكى ترسم داخل هذه المساحة، استخدم خاصية الـ Canvas للـ Control .

● TabIndex: هذا يساوى الـ TabSheet Index الذى يكون الصفحة، بصفر للصفحة الأولى، واحد للثانية، وهكذا .

● Rect: القيم الموجودة فى هذا السجل تحدد الأحداث المتعلقة للرسم فى الـ tab . غالباً، يمكنك استدعاء الـ methods فى الـ Canvas وتقرير الـ Rect أو قيمة داخلية مثل الـ Rect.Left والـ Rect.Top على أنها arguments .

● Active: يكون محدد بـ True اذا كان الـ tab للصفحة الذى يتم رسمه هو للصفحة النشطة (page tab الذى يتم إختياره فى وقت التشغيل) . اذا كان محدد بـ False، يكون الـ procedure قد رسم صفحة غير نشطة - ألا وهى، صفحة ذات controls غير مرئية .

يستخدم البرنامج هذه الـ parameters لرسم أيقونات ونص فى كل page tab . من خلال عبارة الـ with نلاحظ أننا نستخدم خاصية الـ Canvas المرتبطة بالـ page tab وليس الـ object آخر :

```
with Control.Canvas do
begin
  // Call methods for the Control Canvas
end;
```

## الباب الثاني عشر : التعامل من خلال Dialog Boxes

والخطوة الأولى هي لتحديد ما اذا كنا نرسم الصفحة النشطة . اذا كان كذلك ، يحدد البرنامج نمط ال Canvas font بالخط المائل ؛ وإن لم يكن كذلك ، يتم تحديد النمط بال font العادي (لا ثوابت نمط معين) ، باستخدام عبارة ال if :  
if active  
then Font.Style := [fsItalic]  
else Font.Style := [ ];

لعرض تعليقات page tab ، يستدعى البرنامج ال TextRect method في  
: Canvas

```
TextRect(Rect, Rect.Left + 20, Rect.Top + 3,  
PageControl1.Pages[TabIndex].Caption);
```

ال arguments الثلاث الأولى لل TextRect تحدد ال Rect التام ، الذي يعرف الحدود التي يجب أن يظهر فيها النص ، والإحداثيات اليسرى والعلوية لل pixel الأيسر العلوي للنص . وال argument الأخيرة هي النص الذي يجب عرضه . وهذا قد تم أخذه من ال Pages array ل PageControl ، باستخدام قيمة ال TabIndex الذي تم تمريرها ل OnDrawTab . هذا يحدد موضع ال TabSheet الذي يمثل الصفحة وخاصة ال Caption له .

لرسم الأيقونات يستدعى البرنامج ال Draw method في ال ImageList الذي يحمل صورة الأيقونة :

```
ImageList1.Draw(Control.Canvas,  
Rect.Left + 2, Rect.Top + 2, 0);
```

تحدد ال argument الأولى أن ال Canvas الخاص بـ page tab يستخدم في رسم الصورة . وال arguments الآخرتان تحددان موضع الصورة . وال argument الأخيرة هي Index الصورة في ال ImageList . بالرغم من أن التطبيق يرسم نفس الأيقونة في كل page tab ، إلا أنه من السهل عرض أيقونات مختلفة . قم بتحميل صور الأيقونات في ال ImageList الخاص بك ، وحدد قيم الفهرس الخاصة بها بـ ImageList.Draw . (فكرة : يمكنك استخدام ال TabIndex parameter الذي تم تمريره ل OnDrawTab - بهذه الطريقة ، تصبح قيم فهرس الصورة تماماً مثل ال TabSheet في ال PageControl) .



**ملحوظة:** إن عروض page tabs يتم تحديدها بصورة تلقائية بكم النص الذى يعرضه OnDrawTab.

Nota

## ١١. TabControl component:

إن الطريقة السهلة لإنشاء controls dialogs أخرى متعددة الصفحة هي باستخدام الTabControl على الWin32. وعلى عكس الPageControl الأكثر تعقيداً، إن الTabControl ليس لها صفحات منفصلة، بالرغم من أنها تبدو كذلك. إن الصفحات فى الTabControl هي مجرد خداع. يستطيع المستخدمون إختيار page tabs الTabControl، كما يفعلون مع الPageControl- من الناحية البصرية، يبدو الobjects وكأنهما واحد. ولكن، لأن الTabControl ليس له صفحات حقيقية.

وإحدى الاستخدامات العملية للTabControl هي إنشاء forms إدخال بيانات ذات مجموعات من الcontrols تتغير محتوياتها اعتماداً على الtab المختار. على سبيل المثال، قم بتحميل ملف مشروع الPageTab.dpr فى Delphi- على القرص المدمج، يوجد هذا الملف فى دليل الSource\PageTab. والTabControl الموجود على اليمين [راجع شكل (١٢-٧)] يوضح ثلاثة Edit objects. قم بتشغيل البرنامج بضغط F9، واختر كلاً من page tabs الأربع تبقى الEdit controls كما هي، ولكن تتغير محتوياتها اعتماداً على الصفحة النشطة. ولكن تذكر أنه لا يوجد صفحات حقيقية فى هذه الحالة- page tabs فقط تغير مظهرها لتشير الى أى مساحة فى الcontrol يتم إختيارها.

لإدخال نص فى الEdit، ينفذ البرنامج هذا الprocedure:

```
procedure TMainForm.TabControl1Change(Sender: TObject);
var
  S: String;
begin
  S := IntToStr.TabControl1.TabIndex + 1);
  Edit1.Text := 'Edit1 Tab ' + S;
  Edit2.Text := 'Edit2 Tab ' + S;
  Edit3.Text := 'Edit3 Tab ' + S;
end;
```

## الباب الثاني عشر : التعامل من خلال ال Dialog Boxes

هذا ال code لأغراض العرض فقط ، ولكنها تظهر كيفية تعديل ال TabControl اعتماداً على أى page tab يتم إختياره . إن ال OnChange لل TabControl ، والذي يتم استدعاؤه عندما يختار المستخدم page tab . لتحديد ال tab الذى تم إختياره ، استخدم خاصية ال TabIndex لل TabControl . ال tab الأول يحمل رقم صفر ، والثانى رقم واحد وهكذا . لترقيم ال labels الموضحة فى ال Edit ، يضيف البرنامج واحد لهذه القيمة وينشئ S string لإضافته لل static text الموضحة . وهذه يتم تعيينها لخصائص ال Text لل Edit . فى برنامج حقيقى ، يمكنك استخدام تقنيات مشابهة لتغيير محتويات ال controls - بقراءة معلومات النص من الملف ، مثلاً .

**Tip** **فكرة:** فى وقت التصميم ، يمكنك تغيير ال TabIndex التابعة لل TabControl لتنشيط tab مختلفاً . على عكس ال PageControl ، لا تعد ال tabs ال TabControl قابلة للتعامل معها فى وقت التصميم .

### خصائص ال TabControl:

إن خصائص ال TabControl تشبه خصائص ال PageControl ، لذا لن أذكرها منفصلة (أنظر خصائص ال PageControl فى هذا الباب) . إن الخاصية الوحيدة المختلفة هى ال Tabs ، وهى TStrings . اضغط قيمة هذه الخاصية مرتين ، أو اضغط الزر البيضاوى لها ، لفتح Delphi string-list editor . كل سطر نص فى هذا ال editor ويمثل Page tab label . اكتب أى عدد من ال labels التى تحتاجها لإنشاء ال page tabs ال TabControl .

### تحديد حجم النافذة:

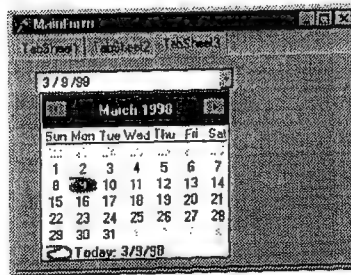
فى النسخ السابقة من ال Delphi لم يكن فى الإمكان تحديد حجم النافذة (أو على الأقل ، لم يكن سهلاً) . لقد كان هذا يؤدى عادة الى مشاكل عندما يقوم المستخدمون بإعادة تحديد حجم النافذة ، مما يغطى بعض ال controls الخاص بالنافذة . والخيار الوحيد أن تجعل حجم النافذة ثابتاً ، ولكن كان هذا معوق للغاية .

إن 4 Delphi يضيف خاصية جديدة لـ TControl class : وهى ال TSizeConstraints . يمكنك استخدام هذه الخاصية لتحديد الحد الأقصى والأدنى

للعروض والإرتفاعات لأغلب ال components المشتقة من ال TControl . حتى نافذة ال from يمكن أن تكون مقيدة الحجم .

يعد استخدام خاصية ال Constraints أمراً بسيطاً . وكمثال ، ابدأ تطبيقاً جديداً وإبحث عن خاصية ال Constraints فى نافذة ال Object Inspector لل Form1 للتطبيق اضغط مرتين علامة الزائد الصغيرة الى اليسار من ال Constraints . هذا فتعرض القيم الخواص الفرعية الأربعة : MaxHeight ، MaxWidth ، MinHeight ، و MinWidth . إن هذه القيم تكون صفراً فى المعتاد ، مما يخلق كل ال Constraints . أدخل أى أعداد صحيحة فى هذه الخصائص لتحديد النافذة أو إرتفاع وعرض أى control آخر . إن القيم هى أعداد صحيحة موجبة تمثل أحجام ال control بال pixels .

وبرنامج العرض Constraint الموجود على القرص المدمج فى دليل ال Source\Constraint يوضح كيفية إنشاء ال PageControl . والهدف فى برنامج العرض هو ضمان أن المستخدمين لا يستطيعون تصغير نافذة dialog الى حد كبير بحيث تؤثر على controls أخرى مثل ال CheckBoxes وال RadioButtons . فى نفس الوقت ، نريد منع ال dialog من أن يملأ الشاشة ، مما يهدد حالة الشاشة الفعلية وتبدو سيئة . يوضح شكل (١٢-١٢) عرض البرنامج - لاحظ أن ال PageControl يملأ النافذة . ان ال PageControl هو الذى تم تقييد حجمه ، ولكن لان النافذة تملك ال control ، فهى الأخرى لها نفس القيود . ان ملف ال Main.pas لهذا البرنامج ليس له code مهمة ، لذا اذكرها هنا .



شكل (١٢-١٢): ان ال PageControl يقيّد حجم نافذته ، مما يضمن ان المستخدمين لا يستطيعون تصغير النافذة وتغطية ال controls وتكبير النافذة اكبر من حجمها العملى .

## الباب الثاني عشر : التعامل من خلال ال Dialog Boxes

**ملحوظة:** أن ضغط زر ال maximize يكبر النافذة لأقصى عرض وطول محدد لها ، وكذلك يضع النافذة إلى الركن الايسر العلوى من الشاشة- قد تريد إضافة ملحوظة فى توثيق تطبيقك حول تأثير احجام النافذة .

Note

لكى تنشئ البرنامج بنفسك ، إتبع مايلى :

١- قم باسقاط PageControl على form خالية . اضغط زر الفأرة الايمن داخل ال PageControl واختر New Page من القائمة لإضافة صفحات قليلة .

٢- قم باسقاط بعض ال controls الاخرى فى صفحات ال PageControl . لا يهم أى ال controls تستخدم . على سبيل المثال ، أضف ثلاثة من ال CheckBoxes الى الصفحة الأولى ، وثلاثة من ال RadioButtons الى الصفحة الثانية ، و DateTimePicker الى الصفحة الثالثة .

٣- مازلنا فى ال Delphi ، حدد حجم النافذة باقصى ما تريد . لاحظ قيم خصائص ال Width وال Height لل PageControl . (تأكد من ان ال Object Inspector يعرض خصائص ال PageControl1- اذا لم يكن ، اختر ذلك من القائمة الخاصة بال Object Inspector) . اضغط مرتين خاصية ال Constraints ، وإدخل قيم ال Width وال Height المشار إليها فى ال MaxWidth وال MaxHeight على التوال .

٤- قم بتصغير النافذة الى اصغير حجم يبدو جيداً ولا تغطى أى من ال controls التى اضفتها لل PageControl أعد الخطوة الثالثة ، ولكن هذه المرة إدخل قيم ال Width وال Height المشار إليها فى ال MaxHeight وال MaxHeight .

٥- قم بتشغيل البرنامج بضغط ال F9 . حاول ان تحدد حجم النافذة- فهى لا تصغر ولا تكبر عن الحدود المحددة . وكذلك ، اضغط زر الشاشة الكاملة ولاحظ ان الشاشة تكبر فقط الى اقصى حد للحجم المسموح به .

**تحذير:** ان إضافة controls الى حاوية مثل ال PageControl يؤدى الى إعادة تحديد قيم خاصية ال Constraints لها بصفر . قم ببرمجة خاصية ال Constraints فى هذه الحالات فقط بعد إدخال كل ال controls الصغيرة فى الحاوية الأم .



في وقت التشغيل ، يمكنك ان تتخطى قيود ال controls باستخدام event handler جديد مخصص لهذا الغرض . قد تفعل هذا لتسمح للمستخدمين بتكبير نافذة إلى حجم الشاشة الكاملة .

**على القرص المدمج:** لتجربة ال event handler هذا ، أعد تحميل مشروع ال Constraint.dpr من دليل ال Source\Constraint على القرص المدمج الى Delphi ، ثم إتبع هذه الخطوات :



- ١- اختر MainForm باستخدام القائمة التابعة لل Object Inspector .
- ٢- اضغط ال Events tab على نافذة ال Object Inspector .
- ٣- اضغط مرتين ال OnConstrainedResize لانشاء event handler خالى . إملأ code هذا البرنامج باستخدام القائمة (١٢-١١) .
- ٤- قم بتشغيل البرنامج بضغط F9 . أعد تحديد حجم النافذة- يمكن ان تكبر الآن الى مساحة الشاشة كاملة . كذلك حاول ان تضغط زر الشاشة الكاملة للنافذة ، الذى يعمل الآن بشكل طبيعى .

**القائمة (١٢-١١) :** أدخل ال code من OnConstrainedResize هذا الى مشروع ال Constraint لتوضح كيفية تخطى قيود النافذة

```
procedure TMainForm.FormConstrainedResize(Sender: TObject;
var MinWidth, MinHeight, MaxWidth, MaxHeight: Integer);
begin
    MaxWidth := Screen.Width;
    MaxHeight := Screen.Height;
end;
```

هذا ال event خاص لل TPanel classes ، و TScrollBar ، و TPanel . استخدمه لتغير فى وقت التشغيل قيم خاصية ال Constraint التى تم إدخالها فى وقت التصميم . على سبيل المثال ، كما هو موضح هنا ، يمكنك تحديد قيم ال MaxWidth وال MaxHeight على التوال للcontrols لخصائص ال Width وال Height لل Screen object ، مما يتيح للنافذة ان تتماثل فى الحجم حجم الشاشة التامة .

## إنشاء الـ Docking Controls:

كما تعرف غالباً يمكنك وصل نوافذ Delphi 4 المتنوعة معاً. على سبيل المثال، اختر View|Project Manager، ثم اضغط واسحب نافذة الى الـ code editor عند الحد الايمن أو الايسر. عندما يغير خط السحب الحجم، اطلق زر الفأرة وتتصل النافذة بالـ code editor. يعتبر هذا مفيداً بشكل خاص مع عروض التقنيات العليا، حيث يوجد وفرة في الاماكن الى اليمين أو اليسار من مساحة تحرير النص. لفك وصل نافذة، اضغطها واسحبها حتى يتغير حد تمييزها مرة اخرى ليعود لحجمه الطبيعي، تم اطلق الفأرة.

يمكنك ايضاً إنشاء docking controls الخاصة بك باستخدام Delphi-النسخ الأولية ليس بها هذه الميزة. لتفعل هذا، إنك تحتاج الى objects اساسيين:

● الـ docking site object.

● الـ dockable control object.

و dockable control object يمكن ان يكون أى component له خاصية DockSite. هذا يشمل الـ Form، الـ Panel، الـ PageControl، الـ TabControl، الـ ToolBar، الـ CoolBar و components اخرى. يجب ان يكون.

الـ docking site object قادراً على ان يملك controls اخرى الـ components البسيطة مثل الـ Buttons والـ CheckBoxes لا يمكن ان تكون docking sites.

و dockable control object هو أى component له خاصيتين هما: DragKind و DragMode. قليل من الـ components فقط هي التي لها DragMode- ولكن لكي تكون dockable، يجب ان يملك الـ control الخاصيتين. وامثلة الـ components التي لا يمكن ان تكون dockable هي تلك الموجودة على الـ Win3.1، وتلك التي ليس لها أى شكل معين عند إسقاطها على الـ form مثل الـ Timer و الـ MainMenu.

Tip . فكرة: ان اغلب التطبيقات تستخدم الـ Panel كـ docking sites. ان الـ

Dockable controls هي تلك التي تحمل الـ controls الأخرى مثل الـ

ToolBar، الـ CoolBars، والـ PageControls.

ان النوافذ ال Dockable اصبحت ممكنة بواسطة التعديلات التى تمت فى Delphi 4 للـ TControl والـ TWinControl . والـ docking sites يجب أن تكون object لـ class تنحدر من TWinControl . يجب ان يكون الـ dockable control يجب أن يكون object لـ class مشتقة من الـ TControl . تنحدر الـ TWinControl class نفسها من الـ TControl " لذلك ، فإن كثير من الـ docking site يمكن أيضاً أن تكون نوافذ docking . على سبيل المثال ، الـ ToolBar ، يمكن ان يكون موقع docking site لنوافذ اخرى ، أو يمكن أن يكون docking لنفسه . من الممكن أيضاً ان يقوم بالعملية-ك docking site لحمل نوافذ اخرى ، و dockable control يمكن للمستخدمين سحبه لإلحاقه بالـ form .

### تكوين الـ docking site:

يمكنك انشاء docking site لأى control بواسطة خاصية الـ DockSite . حدد هذه الخاصية بـ True لتسمح للـ control ان يقبل نافذة اخرى (تسمى docked client) . الـ DockSite عبارة عن ، CoolBar أو Panel أو Form ، ولكن ان يكون أى حاوية اخرى لديها خاصية الـ DockSite .

فى اغلب الحالات ، انك تريد أيضاً ان تحدد خاصية الـ AutoSize للـ DockSite بـ True . عندما تكون الـ DockSite و الـ AutoSize محددتان بـ True ، لا يكون الـ control مرئياً فى وقت التشغيل حتى يتم وصل control آخر به . فى العادى ، هذا هو التأثير المطلوب- ان المستخدم النهائى يرى فقط الخط الخارجى لحد تمييز للـ docked control ، وليس الـ docking site الفعلى .

### إنشاء dockable object:

كما ذكرنا ، يمكن لـ dockable control object ان يكون أى component له خاصيتى الـ DragKind والـ DragMode . على سبيل المثال ، يمكن ان يصبح الـ ToolBar عبارة عن dockable control . لتفعل هذا ، حدد خاصية الـ DragKind للـ ToolBar بـ dkDock والـ DragMode بـ dmAutomatic . وللتحكم فى docking control (والذى لا يعد ضرورياً) ، يمكنك تحديد الـ DragKind بـ dkDock (الافتراضى) وتنفيذ OnDragOver ،

## الباب الثاني عشر : التعامل من خلال ال Dialog Boxes

OnEndDrag، و OnDragDrop . اذا كان ال DragMode محدد بـ dmManual ، استدع ال BeginDrag لتبدأ عملية السحب .

لتحدد نوع ال object الذى يمكن ان يحتوى على dockable control عندما يتم فصله عن docking site ، حدد خاصية ال FloatingDockSiteClass ال TMainForm ، أو الى أى control آخر . ولكن ، اذا كان ال dockable control منحدر من ال TWinControl ، فهذه الخطوة غير مطلوبة لان الفصل سيكون بصورة تلقائية .

### افكار للمستخدم الخبير

• فى الحالات التى يكون فيها المتغيرات المتعددة لـ class غير مذكورة فى ال source code للبرنامج ، يمكنك حذف كل المتغيرات الا واحدة لتوفير اربع بايت لكل متغير فى ال class objects . على سبيل المثال ، افترض انك وضعت ٢٤ TSpeedButton على form ، ولكنك لم تذكر أى منها بوضوح فى ال source code للبرنامج . يمكنك حذف تعريفات الزر من ال form class - هذا لا يحذفهم من ملف ال .dfm . لل form - وتوفر اربعة بايت لكل حذف . يجب ان تترك تعريفاً واحداً على الأقل من كل نوع من المتغيرات بحيث يتم ربط ال code VCL المتعلقة بملف ال exe code . النهائى .

• ان ال Modeless dialogs الموجودة على الشاشة اثناء العمليات الطويلة قد لا تتفاعل بشكل كاف لمدخلات المستخدم . هذا يعتبر حذراً من حدود ال Windows الذى يمكنك دائماً ان يمكن أن تتعامل معه من خلال loops باستدعاء ال ProcessMessages لل Application العام . على سبيل المثال ، اكتب time-intensive loops مثل هذه :

```
while Flag do
begin
    PerformOperation;
    Application.ProcessMessages;
    Flag := ContinueOperation;
end;
```



• إذا كنت على دراية بال Borland Pascal ، فقد تكون كتبت code لتنقل معلومات dialog من وإلى متغيرات التطبيق. فى Delphi ، قم بتعريف هذه المتغيرات ، أو توصل إليهم مباشرة من خلال ال class public section . على سبيل المثال ، إتبع هذه الخطوات لتوضيح كيفية الوصول لـ dialog controls :

١- أضف Button على form (module Unit1) . اختر أمر أو New SpeedButton واختر مربع ال Standard dialog ذى الازرار على طول الحد الايمن .

٢- أضف ال CheckBox على ال form الثانية (module Unit2) .

٣- أضف ال Unit2 لتعريف ال uses لـ Unit1 . هذا يسمح لل event handlers لل Button ، ول code اخرى فى ال form الرئيسية ، بالوصول الى العناصر العامة لل module الثانية .

٤- قم بتنفيذ OnClick لل Button فى ال module Unit1 كما يلى . كما توضح العبارة ، يمكنك الوصول مباشرة لخاصية ال Checked فى ال CheckBox1 لل dialog . يخزن ال dialog object قيم ال control الخاص به ، ولذلك ، فإن آلية النقل التابعة لل Borland Pascal لا حاجة إليها فى تطبيقات Delphi :

```
with BtnRightDlg do
if ShowModal = mrOk then
if CheckBox1.Checked then
    ShowMessage('Checkbox is enabled')
else
    ShowMessage('Checkbox is disabled');
```

• فى النسخ السابقة من Delphi ، تلقيت ال GPF (خطأ الحماية العام) مباشرة عند تشغيل تطبيق اذا اشرت الى method فى form لم توجد بعد . على سبيل المثال ، فى برنامج يحتوى على dialog box ، لا يمكنك استدعاء YourDialog.AnyMethod فى ال OnCreate لل form الرئيسية لان YourDialog لم يتم إنشائه بعد . اذا كنت لا تزال تستخدم نسخة أولية من Delphi ، لإصلاح البرنامج ، إنقل العبارة لـ OnActivate لل form واستخدم ال Boolean flag لتشير اذا ما كان ال dialog قد تم إنشائه . هذه المشكلة لا تظهر فى Delphi 3 والنسخ اللاحق .

## الباب الثاني عشر : التعامل من خلال الـ Dialog Boxes

• كل صفحة في الـ TabbedNotebook تعتبر TTabPage. إرجع الى هذه الـ objects من خلال خصائص الـ Objects والـ Pages. على سبيل المثال، يعرض الـ procedure التالي الـ Caption لباب الصفحة الأولى في الـ TabbedNotebook:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  P: TTabPage; { Reference to a TabbedNotebook page }
begin
  with TabbedNotebook1 do
    P := TTabPage(Pages.Objects[0]); { P refers to page
    1 }
    ShowMessage(P.Caption); { Display page tab label }
end;
```

• لتمنع المستخدمين من اختيار صفحة معينة في الـ TabbedNotebook، قم بإنشاء الـ OnChange event، الذي له اثنين من الـ parameters بالإضافة للـ Sender: NewTab، وهو يمثل الـ tab الذي يضغطه المستخدم؛ والـ AllowChange، الذي يمكنك تحديده لتشير اذا ما كانت هذه الصفحة قابلة للاختيار. على سبيل المثال، في الـ OnChange للـ TabbedNotebook استخدم هذه العبارة لإبطال الصفحة الثالثة (فهرس الصفحة الأولى يساوى صفراً):

```
if NewTab = 2 then
```

```
  AllowChange := False;
```



### المشروعات التي يمكنك تجربتها

(١٢-١): اكتب telephone dialer dialog. يجب ان تبدو النافذة مثل هاتف الصوت. ادخل رقم هاتف من خلال الـ ShowModal اذا ضغط المستخدمون زر الـ Dial.

(١٢-٢): أضف الـ ColorDialog وأمر الـ SysColor من الباب السابق.

(١٢-٣): متقدم: نفذ أوامر الـ Find والـ Replace للـ TStringList.

(١٢-٤): متقدم: استخدم ال Notebook وال TabSet لإنشاء Options dialog متعدد الصفحات. هذا يتطلب مجهوداً زائداً عن استخدام ال TabbedNotebook كما هو مقترح في هذا الباب، ولكنه يعطيك مرونة أكثر. على سبيل المثال، يمكنك وضع ال TabSet في المكان الذي تريده، وإذا كانت المسافة محدودة، يمكنك استغلال امكانية ال TabSet على تحريك tab labels أفقياً.

### ملخص:

- يعتبر ال dialog box أذن وصوت البرنامج. في Delphi، اية form يمكن ان يكون لها صفات ال dialog لخص المستخدمين، على عرض رسائل، وتقديم اختيارات.
- يحتوى ال Delphi ال Windows dialogs العامة في ال ColorDialog، ال FindDialog، ال FontDialog، وال OpenFileDialog، ال ReplaceDialog، وال SaveDialog. يشرح أيضاً هذا الباب ال TabSet، ال Notebook، وال TabbedNotebook المرتبطة، والتي تعتبر مرنة في انشاء dialog box متعدد الصفحة.
- يمكن لل Dialogs أن يكون modal أو modeless (تمسك به، ولكن لا تحافظ عليه). استدع ال ShowModal method لل form لعرض نافذة ال modal. واستدع ال Show لعرض نافذة ال modeless dialog. لعرض Windows dialogs عامة، استدع ال Execute methods الخاصة لل component.
- إنك تستخدم في المعتاد ال TabSet، وال Notebook معاً لإنشاء toolbar تشبه ال VCL palette الخاصة بـ Delphi. يوضح تطبيق ال Palette الخاص بهذا الباب هذه التقنية.
- ان الفرق الاساسي بين زوج من ال TabSet، وال Notebook، وال TabbedNotebook، هو المظهر. تبدو ال TabbedNotebooks مثل

## الباب الثاني عشر : التعامل من خلال Dialog Boxes

stacked file folders ذات tabs عليها labeled بطول القمة. إن ال dialogs TabbedNotebooks مناسب بشكل خاص فى إنشاء dialogs اختيارات متعددة الصفحة .

● إن كتابة أوامر ال Find and Replace ليست مهمة سهلة . توفر ال FindDialog وال ReplaceDialog الخاصة بـ Delphi الوسليه للبحث والإبدال modeless ، ان تطبيق ال FindRepl الخاص بهذا الباب يضع الخطوات الرئيسية .

● ان برنامج العرض TextDemo الخاص بـ Delphi (انظر ملف ال Search.pas فى دليل ال Demos\Textdemo) يحتوى على نظام بحث للنص والذي يمكنك اخذه لعمليات ال Find وال Replace .

● ان ال PageControl الحديث نسبياً ييسط عملية إنشاء dialogs متعددة الصفحة ، كما هو معروض بواسطة أمر ال Options الخاص بتطبيق تعتبر كل صفحة من ال PageControl هى TabSheet ، والذي يحمل controls مثل ال Buttons وال CheckBoxes المرتبطة بهذه الصفحة .

● ان ال TabControl يشبه ال PageControl فى انه يعرض page tabs يمكن للمستخدمين اختيارها . ولكن ، يعتبر ال TabControl شكلاً فردياً ، وصفحاته ما هى الاخداع . والاستخدام الامثل للـ TabControl هو إنشاء شاشات إدخال بيانات مع مجموعات من ال Edit التى تتغير محتوياتها اعتماداً على ال page tab .

● استخدم خاصية ال Constraints الجديدة لتحديد طول وعرض أى control . نفذ OnConstrainedResize للـ Form أو ال Panel أو ال ScrollBox .

● مع Delphi 4 ، يمكنك انشاء docking windows الخاصة بك . أى مثل ال Panel له خاصية ال DockSite يمكن ان يصبح docking site . حدد هذه الخاصية بـ True وحدد أيضاً ال AutoSize غالباً بـ True حتى يكون ال docking site غير مرئى فى وقت التشغيل .

● لإنشاء dockable control، حدد خاصية ال DragKind له بـ dkDock وكذلك حدد ال DragMode له بـ dmAutomatic. فى وقت التشغيل، يستطيع المستخدمون ان يضغوا ويسحبوا ال control الى docking site، ويمكنهم سحبه بعيداً عن docking site لفصل ال control. هذا الباب يلخص فحص الجزء الثانى فى تقنيات إنشاء واجهة التطبيق الخاصة بالمستخدم باستخدام ال Delphi components. فى الجزء الثالث، سوف ترى الجانب الآخر من عملية التطوير - المهام الداخلية التى يؤديها تطبيقك.

# الجزء الثالث

## التطبيق

### محتويات هذا الجزء:

- الباب الثالث عشر: تطوير تطبيقات الجرافيك.
- الباب الرابع عشر: تطوير تطبيقات الطباعة.
- الباب الخامس عشر: تطوير تطبيقات الـ MDI.
- الباب السادس عشر: التطوير مع الـ Clipboard، الـ DDE، والـ OLE.
- الباب السابع عشر: تطوير تطبيقات قاعدة البيانات.
- الباب الثامن عشر: تطوير الـ Charts والتقارير.

إن تصميم واجهة تطبيق المستخدم لبرنامجك، وهى مادة هذا الكتاب حتى الآن، هى مجرد نصف لعبة كتابة برمجيات ناجحة. ويشمل النصف الثانى مميزات برنامجك - الا وهى ما يقوم به مما يجعله فريداً عن البرامج الأخرى. فى هذا الجزء، سوف تتعلم تقنيات لتطوير تطبيقات تستغل العناصر التى عرفتها من الجزء الاول والثانى.

ان اغلب التطبيقات تستخدم عناصر هامة من المواد الموضحة فى ابواب هذا الجزء. قبل قراءة الابواب القادمة، يجب ان يكون قد اعتدت نوافذ واوامر Delphi، ويجب ان تكون مستوعباً تماماً لتقنيات برمجة واجهة التطبيق الخاصة بالمستخدم وبشكل خاص الـ events وخصائص الـ component - كما هى موضع فى الجزء الثانى.

فى الابواب التالية، سوف تعرف كيفية إنشاء انواع عديدة مختلفة من التطبيقات التى تستخدم الـ components رفيعة المستوى للرسوم الجرافيكية،

الصور المتحركة، مخرجات الطابعة (MDI)، نقل بيانات من خلال ال clipboard، و (DDE)، وال (OLE) object، و ActiveX، و charts وتقارير، وإدارة قاعدة البيانات. بالرغم من ان تطبيقك فريد، بالطبع، الا انه يقع تحت أى من هذه الفئات الموضحة فى فصول هذا الباب.

# الباب الثالث عشر

## تطوير تطبيقات الجرافيكية

محتويات هذا الباب:

• Components

• معلومات حول canvas

• الرسم والتلوين

• تقنيات برمجة الرسوم الجرافيكية

• Picture dialogs

• Animations

بغض النظر عن مهاراتك الفنية، مع مساعدة من خاصية ال Canvas ل Delphi، يمكنك رسم وتلوين أى شكل تتخيله. بالإضافة الى هذا، فإن الجرافيك وانواع البيانات وال procedures وال functions الخاصة ب Delphi قد بسطت الى حد كبير مهام البرمجة مثل العمل بال bitmaps، ووقراءة وكتابة الملفات ذات الاحجام الكبيرة.

فى هذا الباب، سوف اقدم تقنيات الجرافيك الخاصة ب Delphi وسأوضح كيفية استخدام خاصية ال Canvas المكشفة، والتي توفرها كثير من ال components. سوف اغطى ايضاً الموضوعات الجرافيكية المتعلقة مثل الصور المتحركة، التعامل مع ملف الجرافيك، (offscreen bitmaps)، dialogs، picture و arag-and-drop objects.



## : Components

فيما يلي قائمة بـ components الجرافيك الخاصة بـ Delphi :

• **Animate** : استخدم هذا component لإضافة نوافذ صور متحركة لل form أو dialog . يمكنك ان تشغل ملفات ال avi . (السمعية البصرية) المعيارية ، كما يمكنك الاختيار من الصور المتحركة المعيارية لل Windows كتلك التي تراها عند نسخ وحذف ملفات باستخدام ال Explorer . ال Palette : Win32 .

• **Image** : استخدم هذا component لإدخال bitmaps ، ايقونات ، وملفات كبيرة (أو metafiles) في form . ان ال Image object يعد غلافاً يحتوى على object جرافيكى فى خاصية ال Picture التابعة له . ال Palette : Additional .

• **MediaPlayer** : استخدم هذا component للوسائط المتعددة ، وهو كامل الميزات لإقامة برمجيات صوتية وجرافيكية متحركة . كما يوضح هذا الباب ، بقليل من ضربات للوحة المفاتيح ، يمكنك إنشاء viewer ملف ال avi video . (المرئية المسموعة) ال System : Palette .

• **SavePictureDialog ، OpenPictureDialog** : يوفر هذان ال dialogs components معيارية لفتح وحفظ ملفات جرافيكية . ال Palette : Dialogs .

• **PaintBox** : استخدم هذا component لإضافة إمكانيات الرسم والتلوين لل components مثل ال Panel التي ليس لها خاصية Canvas . يمكنك ايضاً استخدام ال PaintBox لتحديد الرسم على مساحة مربعة واحدة أو أكثر فى نافذة ال form . ال System : Palette .

• **Shape** : استخدم هذا component لإنشاء object هندسية الشكل مثل المستطيلات وأشكال بيضاوية ممتلئة وغير ممتلئة . اختر ال object الذى تريده بتحديد خاصية ال Shape لل Shape (الخاصية و ال component لهما نفس الاسم) . ال Additional : Palette .

## معلومات حول الـ Canvas:

ان كل العمليات الجرافيكية تحدث فى اطار خاصية الـ Canvas، التى يوفرها العديد من الـ components. على سبيل المثال، ان للـ form خاصية Canvas، وكذلك الـ TBitmap، والـ TComboBox، والـ TListBox، والـ TPaintBox وغيرها. وتمثل خاصية الـ Canvas الـ Windows Graphics Device Interface (GDI)، وهى توفر سطح مستقل لرسم objects جرافيكية على النوافذ. (يغطى الباب التالى طبع الجرافيك والنص). من الجيد ان ترى الـ Canvas على انها سطح يلون عليه برنامجك النص والرسوم الجرافيكية-إنها ليست "سطحاً"، أنما أكثر منها قناة تنساب من خلالها اوامر الجرافيك الى الشاشة، حيث تصبح بصرية.

**ملحوظة:** الى أولئك المعتادون على برمجة الـ Windows، يجب أن الـ Canvas ماهو إلا object-oriented encapsulation للتعامل مع HDC. لإنشاء رسوم جرافيكية، تقوم الـ methods الموجودة فى الـ TCanvas class باستدعاء الـ functions الـ Windows GDI.

Note

## الرسم مع الـ Canvas:

تعتبر خاصية الـ Canvas شاملة للـ Windows GDI. ودائماً الـ Canvas، تعتبر جاهزة للاستخدام. هى object بكل معناه، وتوفر الـ TCanvas class عشرات الخصائص، الـ methods والـ events. بشكل عام، إنك تستخدم الـ Canvas بطريقتين:

- لتشكيل مخرجات جرافيكية بتعيين قيم ألوان، أنماط، fonts، وقيم أخرى لخصائص الـ Canvas.

- لإنتاج objects بصرية باستدعاء الـ methods جرافيكية، بعضها أيضاً يوفر خدمات مساعدة مثل تحديد عرض للشاشة بالـ pixel لـ string.

**ملحوظة:** أنظر الباب التالى لمزيد من المعلومات حول عرض وطباعة نص مع الـ canvas text output method.

Note

إن خاصية ال Canvas غير متاحة في وقت التصميم في نافذة ال Object Inspector . لاستخدام ال Canvas ، يجب أن تكتب عبارات تحدد قيم لخصائص ال Canvas وتستدعي ال Canvas methods . على سبيل المثال ، لرسم مستطيل أزرق ممتلئ بخطوط قطرية صفراء في نافذة ال form ، أدخل هذه العبارات في OnPaint ال form :

```
with Canvas do
begin
  Pen.Color := clBlue;
  Brush.Color := clYellow;
  Brush.Style := bsDiagCross;
  Rectangle(10, 10, 100, 100);
end;
```

لإدخال هذا ال code ، إبدأ تطبيقاً جديداً ، اختر ال Form1 في ال Object Inspector ، اضغط ال Events page tab لهذه النافذة ، واضغط مرتين في حقل القيمة لل OnPaint . أضف السطور السابقة لل event handler الناتج الذي ينشئه Delphi ، ثم اضغط F9 لتشغيل البرنامج .

وهذه الجزئية من البرمجة تستخدم عبارة ال with التي تخبر ال Delphi أن يستخدم الخصائص و ال methods في ال Canvas . ولأن هذه العبارة تظهر في ال TForm1 method ، فإن ال Canvas الخاص بال form هو الذي يتم ذكره ، ولذلك ، تظهر المخرجات في نافذة البرنامج الرئيسية في هذه الحالة . تقوم العبارات الثلاث الأولى بتعيين قيم لإثنين من خصائص ال Canvas : ال Pen و ال Brush ، مما يحدد الخصائص الفرعية لهذه العناصر كما هو موضح . تقوم العبارة الأخيرة باستدعاء ال Canvas method ، وهو ال Rectangle لرسم ال object جرافيكى باستخدام القيم المعينة . لإنشاء أغلب أنواع الجرافيك ، فإنك تستخدم خطوات مشابهة :

- تعيين قيم لخصائص و الخصائص الفرعية لل Canvas .
- استدعاء ال Canvas methods لإنتاج مخرجات بصرية .

### الرسم بال Shapes:

يقدم Delphi طريقة أخرى لإنشاء ال Shapes . فبدلاً من استدعاء ال Canvas methods لرسم ال object في وقت التشغيل ، يمكنك إضافة ال shape على

ال form . إن النتيجة النهائية من الجانب البصرى واحدة ، ولكن ال Shape يعطيك سبل ترتيب ال objects الجرافيكية فى وقت التصميم بالضغط والسحب وبتعيين قيم خاصة . على سبيل المثال ، أضف من ال Additional palette على ال from من الفصل السابق ، أو استخدم ال from جديدة . اختر قيمة خاصة Shape مثل ال stRoundRect ، أو أدخل هذه العبارة فى ال OnCreate أو أى event handler آخر لل form .

```
Shape1.Shape := stRoundRect;
```

لتغيير لون ونمط ال Shape فى وقت التصميم ، استخدم ال Object Inspector لتعيين قيم للخصائص الفرعية لل Pen وال Brush . تؤثر ال Brush على داخل ال object . وتؤثر ال Pen على حد تمييز ال object . جرب مواصفات مختلفة ، أو عين قيم فى وقت التشغيل بعبارات مثل التالية فى ال OnCreate أو فى event handler آخر لل form :

```
with Shape1 do
begin
  Shape := stRoundRect;
  Brush.Color := clLime;
  Brush.Style := bsCross;
  Pen.Color := clNavy;
  Pen.Width := 3;
end;
```

لأن كثير من ال components لها خاصية Canvas ، إذا لم تظهر صور الجرافيك كما هى متوقعة ، قد يكون السبب عبارة ال with التى تشير الى Canvas خاصة بال object وأنت تريد أن تستخدم خواص ال form . أيضاً لا تستخدمها مع عبارة ال with ، أو تستخدم عبارات مثل MainForm.Canvas .

### خصائص ال Canvas:

إن خاصية ال Canvas توفر ثمان خصائص فرعية لتشكيل الجرافيك . يمكنك تعيين قيم الى هذه الخصائص فى وقت التشغيل فقط ، عادة فى ال OnPaint event handler أو ال OnPaint لل form . وفيما يلى وصف مختصر لكل خاصية :

● **Brush**: توفر اللون والنمط داخل حدود الدائرة أو المستطيل أو الشكل الخماسي، وكذلك لون الخلفية للنص. ليس لها تأثير على السطور أو لون واجهة النص. قم بتعيين قيم للخصائص الفرعية للـ **Brush Color** و **Style**.

● **ClipRect**: يضع الصور الجرافيكية داخل حدود هذا المستطيل. إنه مساو عادة في الحجم للـ **client area** بالنافذة. يمكنك تعيين قيم جديدة للقيم الفرعية وهي الـ **Left** والـ **Top** والـ **Right** والـ **Bottom Integer** الخاصة به لوضع المخرجات الجرافيكية في منطقة أخرى. (ملحوظة: أنظر الـ **PaintBox** component للتعامل مع **method** أسهل).

● **CopyMode**: يحدد كيفية تجميع البت عند استدعاء الـ **CopyRect** method للـ **Canvas**، والذي تستطيع استخدامه لنسخ خاصية الـ **Canvas** لإحدى الـ **object** إلى الـ **object** آخر. على سبيل المثال، حدد الـ **CopyMode** بـ **cmNotSourceCopy** لعكس البيكسلات قبل نسخها إلى **Canvas** آخر.

● **Font**: تعيين الخصائص الفرعية لإختيار أنماط الـ **font** للنص الذي ترسمه باستدعاء الـ **TextRect methods** والـ **TextOut** للـ **Canvas**. الـ **Font** هذا لا علاقة له بخاصية الـ **Font** التابعة للـ **form** - حدد خصائص الـ **Canvas.Font** قبل رسم النص.

● **Handle**: من الـ **GDI functions** التي تتطلب **handle**، إلى **HDC**، قم بتمرير **Canvas.Handle** للـ **HDC parameter**. هذا يجعل من الممكن استدعاء الـ **GDI functions** التي لا يحتويها الـ **Canvas**.

● **Pen**: تؤثر على السطور والـ **outlines**، إنك تعين قيم للخصائص الفرعية والتي هي: الـ **Color**، والـ **Style**، والـ **Mode**، والـ **Width** للـ **Pen**.

● **PenPos**: يعطى موضع الـ **pen** الداخلي (**PenPos.X**, **PenPos.Y**) الذي يحدد أين ستظهر المخرجات الجرافيكية التالية. بالرغم من أن **Delphi** يسمح بتعيين قيم جديدة للـ **PenPos**، يجب عليك بدلاً من ذلك استدعاء الـ **MoveTo** method الخاص بالـ **Canvas** لتغيير الموضع الداخلي للـ **Pen**.

● **Pixels:** توفر two-dimensional array لل pixels المنفردة التابعة لل Canvas . يمثل كل عنصر من ال array قيمة ال TColor . على سبيل المثال ، التعبير Pixels[0,0] (استخدم أقواساً مربعة ، يدخل لون ال pixel فى الإحداثية (0,0) . اذا أردت أن تصنع الجرافيك بالطريقة الصعبة) يمكنك أيضاً تلوين ال Pixels الفردية بتعيين قيم لون جديدة لهذا ال array .

### ال events و ال methods ال Canvas:

توفر خاصية ال Canvas عدد من ال methods يمكن أن يساعدك على ترويض الجنون الذى يمر به كل مبرمج ، آجلاً أو عاجلاً ، مع ال Windows GDI . إن العديد من ال shape-Producing methods مثل ال Ellipse ، Arc ، RoundRect و Rectangle ، Polygon ، FloodFill توازى مباشرة ال GDI functions التى تحمل نفس الاسماء . بعد أن تعتاد على التقنية العامة لاستدعاء ال Canvas methods ، فستكون سهلة الاستخدام . على سبيل المثال ، لرسم شكل بيضاوى ، أدخل عبارات مثل هذه فى : ال OnPaint

```
Canvas.Pen.Color := clMaroon;  
Canvas.Brush.Color := clYellow;  
Canvas.Ellipse(10, 20, 100, 100);
```

قم بتعيين قيم لخصائص ال Canvas قبل كل استخدام لخصائص ال Canvas فى ال OnPaint . على سبيل المثال ، لرسم خط أزرق ، عين ال clBlue للخاصية الفرعية لل Canvas Pen.Color ، ثم استدع ال MoveTo methods وال LineTo . لا يمكنك أن تقوم بالتشكيل المسبق لخصائص ال Canvas فى ال OnCreate form لأن ال OnPaint يتلقى ال device context من ال Windows ، وهذه ال devicecontext متوفرة لبرنامجك عن طريق ال Canvas . لذا ، فال Canvas فى ال OnPaint يحتوى أى قيم لخاصية معينة خارج ال event handler .

ال methods الأقل وضوحاً لخاصية ال Canvas والتى يمكنك استدعاؤها تتضمن ما يلى :

● **CopyRect:** ينسخ كل واحدة من خصائص ال Canvas أو جزء منها الى أخرى . حدد خاصية ال CopyMode لتحدد كم pixels تم تجميعه فى النتيجة

النهائية . أنظر ال CopyMode فى ال Delphi online help لمعرفة المواصفات التى يمكنك استخدامها .

● **Draw**: يرسم object التابع لـ TGraphic class ، والتى تعد السلف المباشر للـ TIcon classes والـ TBitmap والـ TMetafile . يمكنك تمرير أى أيقونة أو bitmap أو ملف كبير للـ Draw .

● **DrawFocusRect**: يرسم مستطيل باستخدام عبارات ال OR المنطقية . استدع وقتاً آخر ال arguments المطابقة لمسح المستطيل استخدم هذا ال method لرسم خطوط سحب وإسقاط خارجية .

● **FrameRect**: يرسم مستطيل غير ممتلئ باستخدام ال Pen الخالية ، ولكن متجاهلاً فرشاة ال Canvas استخدم ال FrameRect بدلاً من ال Rectangle عندما تريد outline عادى دون أن تحدد خصائص ال Brush .

● **StretchDraw**: هذه هى نفسها ال Draw ، ولكنها تمد أو تقلص الأيقونة أو bitmap أو الملف الكبير ليتلائم داخل المستطيل المحدد .

خاصية ال Canvas تعرف اثنين من ال events التى قد تكون مفيدة فى احوال خاصة . فى اغلب التطبيقات ، لن تحتاج الى توفير events لهما - فقط كن على دراية بهما اذا ، ما احتجت الى خدماتهما الخاصة . و ال event handlers هما :

● **OnChange**: يتم استدعاؤه بعد ان تتغير قيمة ال Canvas .

● **OnChanging**: يتم استدعاؤه قبل تتغير خاصية ال Canvas مباشرة .

لان خاصية ال Canvas غير متاحة فى وقت التصميم ، لا يمكنك استخدام ال Object Inspector لإنشاء handler لهذين ال event . لاستخدامهما ، قم بتعريف ال procedures فى ال form class :

```
TForm1 = class(TForm)
    procedure MyOnChange(Sender: TObject);
    procedure MyOnChanging(Sender: TObject);
...
end;
```

## الباب الثالث عشر : تطوير تطبيقات الجرافيكية

قم بتعيين اسماء ال procedure لل OnChange وال OnChanging في  
OnCreate الخاص بال form :

```
Canvas.OnChange := MyOnChange;
```

```
Canvas.OnChanging := MyOnChanging;
```

قم بتنفيذ ال procedure لتنفيذ اية اعمال تحتاجها . على سبيل المثال ، اكتب  
ال OnChange كمايلي :

```
procedure TForm1.MyOnChange(Sender: TObject);
```

```
begin
```

```
    SavedPenColor := Canvas.Pen.Color;
```

```
end;
```

**تحذير:** يقوم ال Project Manager الخاص بـ Delphi بحذف  
التعريفات السابقة اذا كان ال event handler لا تحتوي على عبارات أو  
تعليقات عند إنشاء event handler متعددة ، لا تقم بعملية ال  
compile للبرنامج الا بعد إدخال عبارة أو تعليقا واحداً على الأقل في كل  
handler procedure .

## الرسم والتصوير:

تقوم تطبيقات ال Windows برسم shapes منفردة مثل الخطوط  
والمستطيلات الممتلئة . وهى تصور ال objects الجرافيكية لتحافظ على خداع  
النوافذ الواقعة فوق بعضها على الحاسب المكتبى . قد يكون على البرنامج ان يرسم  
ال object مرة واحدة فقط ، ولكنه يجب ان يكون مستعداً دائماً ان يصوره اذا قام  
المستخدم ، مثلاً ، بإخفاء ثم إظهار نافذة التطبيق .

ويظهر الاختيار البسيط الفرق بين الرسم والتصوير . أضف ال Button  
object على ال form . إنشئ ال OnClick لل Button مع البرمجة الموجودة فى  
القائمة (١٣-١) . قم بتشغيل البرنامج واضغط الزر لتحديد ال outline للزر  
بمستطيل احمر . قم بإخفاء وإظهار النافذة ، ويختفى المستطيل .

القائمة (١٣-١): استخدام ال OnClick لعرض الرسم والتصوير

```
procedure TForm1.Button1Click(Sender: TObject);
```



```
var
  X1, Y1, X2, Y2: Integer;
begin
  Canvas.Pen.Color := clRed;
  Canvas.Pen.Width := 2;
  with Button1 do
    begin
      X1 := Left - 3; Y1 := Top - 3;
      X2 := Left + Width + 4; Y2 := Top + Height + 4;
      Canvas.RoundRect(X1, Y1, X2, Y2, 4, 4);
    end;
  end;
```

اترك البرنامج الاختباري وارجع الى Delphi. ادخل OnPaint لل form وانسخ العبارات وتعريف ال var في هذا ال procedure. احذف OnPaint. والآن عندما تقوم بتشغيل البرنامج، فإنه يعطى outline لل Button من البداية، ولا يختفى ال outline عندما تقوم باخفاء وإظهار النافذة.

من الأسهل والأفضل عادة ان تحذف ال event handler بإزالة التعريفات والعبارات الخاصة به، تاركاً ال begin وال end فقط. لا تحذف ال procedure بأكمله، هذا قد يحير مولد ال Delphi code. على سبيل المثال، لإزالة برنامجي OnClick الخاص بال Button، قم بتقليل ال procedure الموجود في القائمة (١٣-١) الى مايلي:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
end;
```

عندما تقوم بعد ذلك بإجراء عملية ال compile لللف ال source code، يحذف ال Project Manager الذكي ل Delphi هيكل وتعريف ال procedure مع أي إشارات الى هذا ال procedure في ال object events.

في اغلب الحالات، يمكنك إضافة عبارات الرسم على ال form أو في OnPaint لل PaintBox. ولكن اذا رسمت ال objects في ال procedures اخرى، يجب ان توفر code ايضاً لإعادة إنشاء تلك ال objects في ال OnPaint. اذا لم

## الباب الثالث عشر : تطوير تطبيقات الجرافيكية

تفعل هذا، فإن (رسوم الجرافيك) الخاصة بك تختفى عندما يخفى المستخدمون نافذة التطبيق ويظهرونها.

قم باستدعاء ال Invalidate method لل form لإخبار ال Windows . ان ال form (أو ال control) تحتاج الى إعادة تصوير . قم باستدعاء ال Update method لل form لإعادة التصوير فوراً . على سبيل المثال، افترض أن برنامجك له ال OnPaint event handler يرسم بعض ال objects الجرافيكية المؤسسة على متغيرات عامة . اذا تغيرت هذه المتغيرات ، لا ترسم ال object الجديد- بدلاً من ذلك ، أدخل هذه العبارات لإعادة تصوير الجرافيك بإجبار ال OnPaint على الحدوث :

```
MainForm.Invalidate;
```

```
MainForm.Update; { optional }
```

لا تفعل هذا أبداً داخل ال OnPaint . إن هدف هذه العبارات هو أن تجعل ال OnPaint يحدث- وتنفذها داخل ال OnPaint تجعل البرنامج يعيد تصوير نفسه الى ما لا نهاية . ويخبر ال Invalidate ال Windows إن ال object يحتاج الى تصوير . يصدر ال Windows رسالة ال wm\_Paint لل object عندما لا يكون هناك رسائل أخرى . وال Update ، والذي يعتبر اختيارياً ، يخبر ال Windows أن يصدر رسالة ال wm\_Paint على الفور .

### ال PaintBox component :

أضف ال PaintBox على form أو حاوية أخرى مثل ال Panel لمنحه إمكانات الجرافيك ، إن كل الأحداثيات متصلة بالحاوية . على سبيل المثال ، أضف ال Panel على ال form وقم بمدها قليلاً لتفسح مكاناً لل PaintBox . اختر ال PaintBox من لوحة ال System واضغط مؤشر الفأرة داخل ال Panel . لإعادة وضع ال object ، ببساطة اسحب ال Panel- لا يجب عليك أن تختار ال PaintBox أيضاً .

لقد أعطيت ال Panel لتوك إمكانات جرافيك جديدة . يمكنك استخدام هذه التقنية لرسم ال objects ، و bitmaps ، و رسوم جرافيكية أخرى في ال toolbar أو ال status ilne . على سبيل المثال ، لرسم مربع أزرق داخل ال Panel ، أدخل

```

event
العبارات التالية فى OnPaint الخاص بالـ PaintBox1 . (تأكد من إنشاء الـ event
handler للـ PaintBox ، وليس للـ form) . ها هو الـ procedure كاملاً:
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  with PaintBox1.Canvas do
    begin
      Pen.Color := clNavy;
      Rectangle(0, 0, 100, 100);
    end;
  end;
end;

```

**ملحوظة:** لاحظ أن عبارة الـ with تشير الى الـ Canvas PaintBox1 . إذا كانت هذه العبارة تشير إلى الـ Canvas وحدها ، فإنها ستشير إلى الـ Canvas الخاص بالـ form بدلاً من ذلك الخاص بالـ PaintBox . كن متنبهاً دائماً إلى أن كل الـ components البصرية لها خصائص Canvas وكن متأكداً من أنك تشير إلى الصحيح منها حتى تذهب مخرجات الجرافيك الخاص بك إلى المكان الذى تريده .

يمكنك أيضاً استخدام الـ PaintBox لقصر الرسم على مستطيل بعينه فى form أو حاوية أخرى . لتقسيم الـ form أو حاوية أخرى إلى قطاعات متعددة ، يمكنك إدخال أى عدد من الـ PaintBoxes .

### استخدام الـ Pens والـ Brushes:

إن كل خاصية Canvas لها خصائص الـ Pen والـ Brush . والـ Pen تحدد لون وخط الخطوط ، وكذلك الـ outlines للـ objects مثل المستطيلات والأشكال البيضاوية . وتحدد الـ Brush أنماط وألوان داخل الـ objects . ولـ Pens أربعة خصائص هامة:

● **Color:** تؤثر على الخط و outline ، ولكن ليس على ألوان داخل الـ object . عين أى قيمة TColor أو ثابت مثل الـ clRed أو الـ clLime للـ Pen.Color . لتتماشى مع ألوان شاشة الحاسب المكتبى ، عين ثابتاً مثل الـ clWindowText أو الـ clWindow .

● **Mode:** تحدد method المنطقي لرسم الخطوط و outlines. عين ثابتاً مثل الـ pmBlack (أسود دائماً) أو الـ pmNotCopy (عكس لون الـ Pen) للـ Pen.Mode. استخدم الـ pmXor بحيث يمكنك إعادة رسم خط لمسحه وتخزين خلفيته الأولى.

● **Style:** اختر نمط خط منقط أو dashed، أو solid. على سبيل المثال، عين ثابتاً مثل psSolid أو psDashDot للـ Pen.Style في الـ Windows ذي الـ 16-bit والـ Windows 95، فإن الخطوط التي يزيد عرضها عن pixel واحد تكون solid دائماً. والـ Windows 95 والـ Windows NT يمكنك من تحديد نمط الخطوط السميكة والرفيعة.

● **Width:** يحدد العرض بالـ pixels للخطوط و outlines. في الـ Windows ذي الـ 16-bit، يجب أن تكون هذه الخاصية مساوية لواحد (الإفترضى) لكي تستخدم Pen Style آخر غير الـ psSolid.

للحصول على قائمة كاملة لشوايت الـ Color والـ Mode والـ Style، أنظر هذه الخصائص الخاصة بالـ TPen class في الـ online help الخاصة بـ Delphi. أو، إذا كان لديك library source code وقت التشغيل الخاصة بـ Delphi، تصفح ملف الـ Graphics.pas.

وللـ Brushes ثلاث خصائص هامة:

● **Bitmap:** يعين TBitmap object ببعد يصل إلى  $8 \times 8$  pixels. تنسخ الفرشاة الـ Bitmap لتتلاءم الـ objects الداخلية.

● **Color:** هذا هو نفسه كما في الـ Pen.

● **Style:** تحدد نمط للاستخدام لتصوير الـ objects الداخلية.

على سبيل المثال، عين ثابتاً مثل الـ bsCross أو الـ bsFDiagonal للـ Brush.Style. استخدم الـ bsSolid عند تعيين الـ Bitmap. لرسم outline مع الخلفية ظاهرة من خلاله، حدد الـ Style بـ bsClear. للحصول على قائمة كاملة بشوايت الـ Style، أنظر خاصية الـ Style للـ TBrush class في الـ online help الخاصة بـ Delphi.

إن الـ Pens والـ Brushes لها methods مثل الـ Create والـ Free والتي تعتبر مشتركة بين كثير من الـ components للـ Brushes والـ Pens أيضاً والـ OnChange والذي يمكنك استخدامه لتخبر البرنامج بأية تغييرات لخصائص الـ Pen والـ Brush، بالرغم من أن استخدام هذا الـ event نادراً ما يكون ضرورياً.

يعتبر تطبيق الـ PolyFlow على القرص المدمج واحداً من أقدم البرامج الإختبارية المعيارية للـ Pascal توجد الملفات فى دليل الـ Source\PolyFlow. يعد الـ PolyFlow مثلاً جيداً على الرسم مقابل التصوير؛ ولأنه يوضح أيضاً كيفية استخدام الـ Timer object لإنشاء صور جرافيكية متحركة. يوضح شكل (١٣-١) عرض الـ Polyflow. القائمة (١٣-٢) تعطى الـ source code للبرنامج.



شكل (١٣-١): الـ Polyflow، والذي يعد برنامج معيارى قديم مع واجهة تطبيق Delphi الحديثة، يوضح الرسم مقابل التلوين، ويوضح كيفية استخدام الـ Timer object لإنشاء صور جرافيكية متحركة

القائمة (١٣-٢): PolyFlow\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
SysUtils, Windows, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, Menus, ExtCtrls;
```

```
const
```

```
maxIndex = 100; { Maximum number of lines visible }
```

```
dx1: Integer = 4; { "Delta" values for controlling }
dy1: Integer = 10; { the animation's personality. }
dx2: Integer = 3;
dy2: Integer = 9;
```

type

```
LineRec = record { Line ends and color }
  X1, Y1, X2, Y2 : Integer;
  Color: TColor;
end;
```

type

```
TMainForm = class(TForm)
  MainMenu1: TMainMenu;
  Demo1: TMenuItem;
  Exit1: TMenuItem;
```

(continued)

Listing 13-2 (continued)

```
  Timer1: TTimer;
  procedure Exit1Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure Timer1Timer(Sender: TObject);
  procedure FormPaint(Sender: TObject);
  procedure FormResize(Sender: TObject);
```

protected

```
  procedure CreateParams(var Params: TCreateParams);
  override;
  private
```

```
  LineArray: array[0 .. maxIndex _ 1] of LineRec;
  Index: Integer; { Index for LineArray }
  Erasing: Boolean; { True if erasing old lines }
  function Sign(N: Integer): Integer;
  procedure InitLineArray;
  procedure MakeNewLine(R: TRect; Index: Integer);
  procedure DrawLine(Index: Integer);
public
```

```

    { Public declarations }
end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.CreateParams(var Params: TCreateParams);
begin
    inherited CreateParams(params);
    with Params.WindowClass do
        { Repaint the window automatically when resized }
        Style := Style or cs_HRedraw or cs_VRedraw;
    end;

    { Return _1 if n < 0 or +1 if n >= 0 }
function TMainForm.Sign(N: Integer): Integer;
begin
    if N < 0 then Sign := _1 else Sign := 1;
end;

{ Erase LineArray and set X1 to -1 as "no line" flag }
procedure TMainForm.InitLineArray;
var
    I: Integer;
begin
    Index := 0;
    Erasing := False;
    FillChar(LineArray, SizeOf(LineArray), 0);
    for I := 0 to maxIndex _ 1 do
        LineArray[I].X1 := _ 1;
    end;
end;

```

```

{ Create new line, direction, and color }
procedure TMainForm.MakeNewLine(R: TRect; Index: Integer);
  procedure NewCoord(var C, Change: Integer; Max: Integer;
    var Color: TColor);
    var
      Temp: Integer;
  begin
    Temp := C + Change;
    if (Temp < 0) or (Temp > Max) then
      begin
        Change := Sign(-Change) * (3 + Random(12));
        repeat
          Color := RGB(Random(256), Random(256), Random(256));
          Color := GetNearestColor(Canvas.Handle,
            Color)
        until Color <> GetBkColor(Canvas.Handle);
        end else
          C := Temp;
        end;
  begin
    with LineArray[Index] do
      begin
        NewCoord(X1, dx1, R.Right, Color);
        NewCoord(Y1, dy1, R.Bottom, Color);
        NewCoord(X2, dx2, R.Right, Color);
        NewCoord(Y2, dy2, R.Bottom, Color)
      end
    end;
  end;

{ Draw or erase a line identified by Index }
procedure TMainForm.DrawLine(Index: Integer);
begin
  with Canvas, LineArray[Index] do
    begin
      Pen.Color := Color;
      MoveTo(X1, Y1);
    end;
  end;
end;

```



```

LineTo(X2, Y2);
end;
end;

{ Draw some lines at each timer interval }
procedure TMainForm.Timer1Timer(Sender: TObject);
var
    R: TRect;
    I, OldIndex: Integer;
begin
    R := GetClientRect;
    for I := 1 to 10 do { 10 = number of lines }
    begin
        OldIndex := Index;
        Inc(Index);
        if Index = maxIndex _ 1 then
        begin
            Index := 0; { Wrap Index around to start }
            Erasing := True; { True until window size
            changes }
        end;
        if Erasing then
        begin
            DrawLine(Index); { Erase old line }
            LineArray[Index] := LineArray[OldIndex];
            MakeNewLine(R, Index);
            DrawLine(Index); { Draw new line }
        end;
    end;
end;

{ Paint or repaint screen using data in LineArray }
procedure TMainForm.FormPaint(Sender: TObject);
var
    I: Integer;
begin
    with Canvas do
        for I := 0 to maxIndex _ 1 do

```

### الباب الثالث عشر : تطوير تطبيقات الجرافيكية

```
if LineArray[I].X1 >= 0 then { Draw non-flagged lines }
    DrawLine(I);
end;

{ Start new lines when window size changes }
procedure TMainForm.FormResize(Sender: TObject);
begin
    InitLineArray; { Erase LineArray and reset globals }
end;

{ Initialize globals and LineArray }
procedure TMainForm.FormCreate(Sender: TObject);
begin
    with Canvas.Pen do
        begin
            Style := psSolid;
            Width := 1;
            Mode := pmXor;
        end;
        Randomize;
        InitLineArray;
    end;

    { End program }
    procedure TMainForm.Exit1Click(Sender: TObject);
    begin
        Close;
    end;

end.
```

إن OnTimer الخاص بال PolyFlow (أنظر ال Timer1Timer) يرسم عشرة خطوط من الصورة المتحركة. ويمكن أن يرسم خطاً واحداً فقط، ولكن هذا سيؤدي إلى إبطاء التأثير بشكل غير مقبول. ويقوم OnPaint الخاص بال form بإعادة رسم النافذة بأكملها بحيث، عندما تريد إخفاءها ثم إظهارها مرة أخرى،

يخرن البرنامج الحالة الحالية للصورة المتحركة . وهذه تقنية Windows جرافيكية نموذجية - تقوم أجزاء من البرنامج برسم الأشكال وتخزين الـ parameters الخاصة بها (أنظر الـ LineArray فى القائمة) . للحفاظ على النافذة، يستخدم OnPaint تلك الـ parameters لإعادة تصوير الجرافيك الخاص بالبرنامج .

حاول تشغيل الـ PolyFlow وقم بتغطية جزء من نافذته . لاحظ أن الـ Timer events تستمر فى التشغيل . بالإضافة الى إنشاء صور متحركة، يمكنك استخدام نفس هذه التقنية لبرمجة عمليات خلفية أخرى، ولكن إنتبه ألا تغلق النظام بتنفيذ عمليات طويلة فى الـ Timer event .

### تقنيات برمجة الجرافيك:

إن كل تطبيقات Delphi الجرافيكية والمؤسسة على الـ form تستخدم وحدة الـ Graphics، والتي توفر classes مثل الـ TPen، TBrush، TFont، TCanvas، وتعريفات متعلقة بها . وهذا الفصل يغطى بعض الـ classes الأخرى والتقنيات الجرافيكية المتقدمة مثل الـ bitmaps والملفات الجرافيكية وكيفية إنشاء الـ objects السحب والإسقاط .

### علاقات class الجرافيك:

عندما تبدأ فى تصميم برنامج جرافيكى، قد تكون غير واثقاً أن تستخدم components أو Shape أو object من الـ TPicture class . لتقرر أفضل مسار لبرنامجك، إفهم أولاً العلاقات الموجودة بين الـ classes التالية :

● **TCanvas:** كما ذكرنا، هذا الـ class توفر سطح يمكنك أن ترسم وتصور عليه الجرافيك باستدعاء الـ methods وتعيين قيم الخصائص . إنك لا تنشئ أبداً TCanvas object مستقل - أنت تستخدم هذا الـ class دائماً كخاصية الـ Canvas لـ object آخر مثل الـ PaintBox أو الـ form .

● **TGraphic:** هذا الـ class هو السلف المباشر للـ TBitmap classes والـ TIcon والـ TMetafile . لا تنشئ أبداً TGraphic object مستقل . بدلاً من ذلك، يمكنك تعيين bitmap object أو أيقونة أو ملف كبير لـ TGraphic parameter أو متغير أو خاصية . وتعمل للـ TGraphic أيضاً كـ class مجردة لكل

## الباب الثالث عشر : تطوير تطبيقات الجرافيكية

أنواع بيانات الصور في Delphi، بما في ذلك لل add-in classes مثل ال JPEG (من وحدة ال JPEG) زائد format ملفات أخرى .

● **TPicture**: هذا ال class تنشئ حاوية تحمل ال TGraphic object، والذي يمكن أن يكون bitmap أو أيقونة أو ملف كبير . إن هذه methods تسمح لك بالتعامل مع الملف الخاصة بال TPicture وتحدد نوع صورة من إمتداد اسم الملف الخاص بها، والذي يبسط عملية التعامل مع ملف الجرافيك . سوف تجد ال TPicture class مستخدمة كخاصية ال Picture في objects أخرى مثل ال Image component، يمكنك أيضاً إنشاء TPicture object مستقل - على سبيل المثال، لتوفر حاوية bitmap تم تحميلها من ملف قرص .

● **TGraphicsObject**: هذا ال class هي السلف المباشر لل TFont classes وال TPen وال TBrush . استخدم ال TGraphicsObject على إنها نوع function parameter أو procedure بحيث يمكنك تمرير Fonts أو pens أو brushes ك arguments للنظم الفرعية . لا تنشئ أبداً object مستقل من class ال TGraphicsObject .

● **TGraphicControl**: تعتبر ال component class هذه هي السلف لـ component classes الجرافيكية مثل ال TSpeedButton، وال TTabButton (المستخدمة بواسطة ال TabbedNotebook)، TShape، TPaintBox، TImage، وال TCustomLabel (السلف المباشر لل TLabel). توفر ال TGraphicControl خاصية Canvas، والتي، على سبيل المثال، يستخدمها ال SpeedButton لعرض glyph bitmap الخاص به . وغالباً لن تحتاج ال TGraphicControl في تطوير التطبيق . إنها مخصصة بشكل خاص لكتابة components بصرية التي ليس لها handles متعلقة بها، ولذلك تأخذ حيزاً أقل في الذاكرة عن ال components التي تعمل كواجهات تطبيق لعناصر ال Windows المعيارية مثل الأزرار ال check boxes . وتعرف ال Controls unit (وليس ال Graphics) ال TGraphicControl .

## الرسم بال components:

لرسم الجرافيك ، يمكنك استدعاء ال Canvas methods وأضف أحد ال components التاليين على ال form أو حاوية أخرى :

● **Image:** هذا ال component يحتوى على Picture object (ال TGraphic class)، والذي يحتوى على خاصية ال Graphic (طبقة ال TGraphic class). لأن ال TGraphic class تعد السلف لل TBitmap class وال TIcon وال TMetafile، فإن ال Image object يمكن أن يحمل ويعرض bitmap أو أيقونة أو ملف كـ بيتر يمكن أن يشار إليه على أنه Image.Picture.Graphic. ويمكن أن يتولى ال Image أيضاً formats ثنائية مثل ال JPEG المتوفرة فى ال JPEG unit وال TJPEGImage class.

● **Shape:** هذا ال component يرسم صورة باستدعاء ال Canvas methods. استخدم ال Shape لرسم وتصوير خطوط ، ومستطيلات ودوائر وobjects أخرى.

ان الفرق بين هذين ال components هو ان ال Image object يحتوى على صور جرافيكية، وال Shape object يرسم ويصور object باستخدام ال Canvas الموروثة. استخدم ال Image components أوال Shape لإنشاء صور جرافيكية فى وقت التصميم استخدم ال Canvas للرسم والتصوير فى وقت التشغيل.

لتخزين objects جرافيكية فى الذاكرة، قم بإنشاء object من ال TPicture class. يمكنك أيضاً إنشاء TBitmap، TIcon، و TMetafile objects منفردة، ولكن من الأفضل دائماً ان تنشئ ال TPicture object الذى يحتوى على Bitmap، أو أيقونة، أو ملف كبير. على سبيل المثال جرب ال OnClick الخاص بال Button فى القائمة (١٣-٣)، والذي يقوم بتحميل وعرض ملف bitmap كاملاً فى وقت التشغيل دون استخدام components جرافيكية.

القائمة (١٢-٢): جرب ال OnClick لهذا ال Button لتحميل

وعرض ملف bitmap كاملاً تحت تحكم البرنامج

```
procedure TForm1.BitBtn1Click(Sender: TObject);
```

```
var
```

```

P: TPicture;
begin
  P := TPicture.Create;
  try
    P.LoadFromFile('C:\Windows\Clouds.bmp');
    Canvas.Draw(0, 0, P.Graphic);
  finally
    P.Free;
  end;
end;

```

توضح القائمة (١٣-٣) بعض تقنيات الجرافيك الأساسية. متغير الـ P يعتبر object من الـ TPicture ، والذي تم إنشاؤه باستدعاء TPicture.Create كالعبارة الأولى للـ procedure. لضمان السيطرة على object الناتج، فإن كل العبارات التي تستخدم الـ P توضع داخل نظام الـ try ويستدعى قالب الـ finally الـ P.Free. هذا يضمن أنه، إذا حدثت إليه exception داخل نظام الـ try، يتم تنفيذ العبارة الموجودة في قالب الـ finally. بغض النظر عن أية أخطاء قد تحدث، فإن الذاكرة المشغولة بالـ TPicture مضمون أنها سوف تمحى بشكل سليم قبل إنتهاء الـ procedure.

داخل الـ try، يقوم الـ LoadFromFile method الخاص بالـ TPicture بقراءة ملف bitmap (غير اسم الملف إذا اردت). لعرض الصورة، يستدعى البرنامج الـ Draw method الخاص بالـ Canvas، والذي يعرفه Delphi كما يلي:

● **X, Y**: إحداثيات Client-area المتعلقة بالـ object والتي توفر الـ Canvas.

حدد هذه القيم بصفر لعرض الصورة في الركن الأيسر العلوي من الـ Canvas.

● **Graphic**: قم بتمرير TBitmap، أو TIcon، أو TMetaFile لهذه parameter. أو، كما توضح القائمة، عند استخدام TPicture، قم بتمرير خاصية الـ Graphic التابعة.

أو، يمكنك إضافة Image على الـ form واستخدام خاصية الـ Picture له لتحميل وعرض. الـ bitmap، البرمجة التالية تنفيذ نفس مهام القائمة (١٣-٣):

```

Picture.LoadFromFile('C:\Windows\Clouds.Bmp');

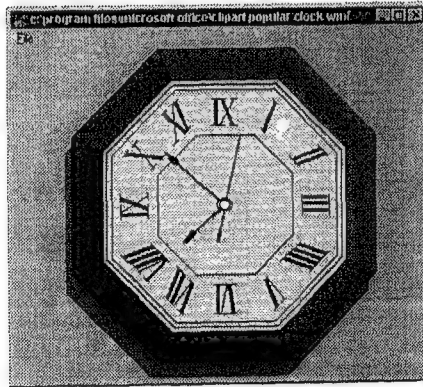
```

ان الفرق بينهما هو مجرد اختيار تصميم . وإن كان من الافضل استخدام component ، فإن ال code السابقة تعتبر الطريقة الأكثر بساطة ، ولا تتطلب منك ان تنشئ وتحرك objects أو تستدعى ال Draw method لل Canvas . اذا كنت تحتاج الى إنشاء واستخدام objects فى وقت التشغيل ، استخدم التقنية الموجودة فى القائمة (١٣-٣) . ولكن ، لا تستخدم ال TPicture ال window handle ؛ بينما تستخدمه ال TImage . وهذا يجعل ال TPicture هى الاختيار الافضل من ناحية الذاكرة .

**ملحوظة:** تعتبر ال class TImage منحذرة من ال TGraphicControl ، وهى لذلك ليس لديها window handle . ولكن ال TDBImage لديها window handle ، وهو طرق بين الاثنين . انظر الباب السابع عشر ، تطوير تطبيقات قاعدة البيانات ، لمزيد من المعلومات حول ال TDBImage و ال controls الأخرى لقاعدة البيانات .

### Metafile, Bitmap, icon files

باستخدام المعلومة السابقة ، يمكنك إنشاء متصفح ملف صورة يمكنه ان يعرض ملف حجم كبير (.wmf) ، أو ايقونة (.ico) ، أو bitmap (.bmp) ، وإنسخ أى من هذه الملفات فى دليل آخر . يوضح شكل (١٣-٢) برنامج ال MetaMore الذى يعرض ملف metafile متوفر مع ال Microsoft Office . والبرنامج ، الموضح فى القائمة (١٣-٤) ، يمكنه أيضاً ان يقرأ ، ويعرض ، وينسخ Icon أو bitmap .



شكل (١٣-٢): يستطيع ال MetaMore عرض أى metafile أو bitmap أو ايقونة

القائمة (١٢-٤) MetaMoreMain.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, Menus, ExtCtrls;
type
  TMainForm = class(TForm)
    Image1: TImage;
    MainMenu1: TMainMenu;
    FileMenu: TMenuItem;
    FileOpen: TMenuItem;
    N1: TMenuItem;
    FileExit: TMenuItem;
    OpenFileDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    FileSaveAs: TMenuItem;
    procedure FileOpenClick(Sender: TObject);
    procedure FileSaveAsClick(Sender: TObject);
    procedure FileExitClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FileOpenClick(Sender: TObject);
```



```

begin
  with OpenFileDialog1 do
    begin
      if Execute then
        Image1.Picture.LoadFromFile(Filename);
        Caption := Lowercase(Filename);
      end;
    end;
  procedure TMainForm.FileSaveAsClick(Sender: TObject);
  begin
    with SaveDialog1 do
      begin
        Filename := Caption;
        if Execute then
          Image1.Picture.SaveToFile(Filename);
          Caption := Lowercase(Filename);
        end;
      end;
    end;

    procedure TMainForm.FileExitClick(Sender: TObject);
    begin
      Close;
    end;
  end.

```

**تحذير:** لا يستطيع الـ MetaMore ترجمة انواع الملفات . على سبيل المثال ، اذا قمت بتحميل bitmap ، يجب ان تحفظها bitmap .



يستخدم الـ MetaMore الـ Image component object ، الذى يعتبر دائماً اسهل الطرق . وبالتبادل ، يمكنك إنشاء Picture object كما توضح القائمة (١٣-٣) . أو ، يمكنك إنشاء object لنوع معين من ملف الصورة ، على سبيل المثال ، توضح القائمة (١٣-٥) OnClick للـ Button الذى ينشئ الـ TMetaFile ويحمل ويعرض metafile .

## الباب الثالث عشر : تطوير تطبيقات الجرافيكية

إذا كان لديك Microsoft Office، يمكنك ان تجد عينة metafile في مسار ال  
Program Files\Microsoft Office\Clipart\Popular ومسارات دليل أخرى).

كما هو الحال مع ال TPicture class، هذه التقنية نحتاج إلى برمجة أكثر  
من استخدام ال Image، ولكنها تستخدم ذاكرة أقل Windows Resource أقل.  
أولاً قم بإنشاء ال TMetaFile باستدعاء ال Create method لل class استخدم لل  
object الناتج في ال try، وحدده في قالب ال finally. هذا يضمن إزالة ال object  
من الذاكرة في حالة حدوث exception.

القائمة (١٣-٥)، جرب OnClick لهذا ال Button

لإنشاء TMetaFile وتحميل وعرض metafile

```
procedure TForm1.Button1Click(Sender: TObject);  
var  
    MetaFile: TMetaFile;  
begin  
    MetaFile := TMetaFile.Create;  
    try  
        MetaFile.LoadFromFile(o^C:\msoffice\clipart\anchor.wmf);  
        Canvas.Draw(0, 0, MetaFile);  
    finally  
        MetaFile.Free;  
    end;  
end;
```

لتحميل metafile، استدع ال LoadFromFile method الخاص بال  
MetaFile (غير اسم الملف إذا لزم الأمر). استدع ال Draw method الخاص بال  
Canvas لعرض ال MetaFile. يمكنك تمرير MetaFile لل Draw لأن ال  
TMetaFile class تنحدر من ال TGraphic.

يمكنك استخدام نفس البرمجة الموجودة في القائمة (١٣-٥)  
لتحميل وعرض ملفات ايقونات و bitmap. قم بإبدال ال MetaFile  
ال TBitmap أو ال TIcon، وقم بتحميل النوع المناسب. استخدم ال TPicture  
[انظر القائمة (١٣-٣)] لتحميل ملفات الايقونة، أو bitmap، أو ال metafile  
حسب امتداد اسم الملف.

بدلاً من ال Draw ، استدع ال StretchDraw لإعادة تحديد حجم صورة لتناسب داخل مستطيل محدد . على سبيل المثال ، للملء client area النافذة بصورة MetaFile ، استبدل عبارة ال Canvas.Draw في القائمة (١٣-٥) بهذا السطر :  
Canvas.StretchDraw(ClientRect, MetaFile);

يعرف Delphi ال StretchDraw على انه :

procedure StretchDraw(const Rect: TRect; Graphic: Tgraphic);

● **Rect** : يحدد مساحة مستطيلة لعرض صورة . قم بتعيين قيم للاعداد الصحيحة Left ، Top ، Right ، Bottom لهذا السجل ، أو ، يمكنك تعيين سجلات TPoint لأعضاء ال BottomRight وال TopLeft التابعين لل Rect .

● **Graphic** : أى TIcon ، أو TBitmap ، أو TMetaFile . ال StretchDraw تعيد تحديد حجم الصورة لتناسب داخل المساحة المحددة لل Rect .

### Bitmap Resources

على عكس ما يحدث فى برمجة ال Windows التقليدية ، فإنك لن تستخدم Resources objects عديدة فى تطبيقات Delphi . وأحد ال exception لهذه القاعدة هو ال bitmap التى تريد ضمنه لتطبيقاتك . يمكنك توزيع ملف bitmap منفصل وتستخدم ال TBitmap أو ال TPicture لعرضه . أو ، يمكنك إنشاء bitmap Resource ك (res) باستخدام ال Image Editor الخاص بـ Delphi على قائمة ال Tools .

بعد إنشاء ملف ال Resource مع bitmap (ولنفترض انك سميتـه YOURBITS) ، اربط Resource فى ملف ال exe code . بادخال هذا الأمر فى تنفيذ ال module :

{ \$R Yourbits.Res }

استخدم البرمجة الموجودة فى القائمة (١٣-٦) لإنشاء bitmap object ، واستدع ال LoadFromResourceName الخاص بال TBitmap . قم بتمرير ال handle الحالى للتطبيق (HInstance) واسم ال resource الذى سيتم تحميله .

## الباب الثالث عشر : تطوير تطبيقات الجرافيكية

وقد يكون bitmap ذات ٢٥٦ لوناً. ويعتبر الـ string YOURBITS هو اسم الـ resource الذى قمت بتعيينه فى الـ Image Editor. (ويمكن ان يكون للـ ملف الـ Yourbits.res اسماً آخر- فلا يجب ان يكون هو نفسه اسم الـ resource الذى يحتويه). استدع الـ Draw methods أو الـ StretchDraw كما هو موضح فى القائمة.

```
القائمة (١٣-٦): الـ Button يوضح كيفية تحميل وعرض الـ bitmap
resource المرتبط بملف code exe. للبرنامج بواسطة الـ Image Editor الخاص بـ Delphi
procedure TForm1.Button1Click(Sender: TObject);
var
  YourBits: TBitmap;
begin
  YourBits := TBitmap.Create;
  try
    YourBits.LoadFromResourceName(HInstance,'YOURBITS');
    Canvas.Draw(0, 0, YourBits);
    (* Canvas.StretchDraw(ClientRect, YourBits); *)
  finally
    YourBits.Free;
  end;
end;
```

### : offscreen bitmap

توفر الـ TBitmap خاصية Canvas التى يمكنك استخدامها. قم بإنشاء TBitmap ، واستدع الـ methods فى خاصية الـ Canvas التابعة لرسم الـ off Screen له للرسم على الـ Canvas لكى يبقى مخفياً عن الانظار. يمكنك عندئذ عرض الجرافيك الناتج مرة واحدة باستدعاء الـ methods مثل الـ Draw، أو الـ StretchDraw، الـ CopyRect، والـ BrushCopy لـ Canvas تابع لـ form أو الـ PaintBox. هذه التقنية تعتبر هامة فى الصور المتحركة، وكذلك لإخفاء تفاصيل تكوين صورة معقدة حتى يكتمل الشكل.

توضح القائمة (١٣-٧) التقنيات الاساسية المطلوبة للرسم فى الـ bitmap offscreen. لتجربة الـ code استخدم القائمة كأنها الـ Button onClick. يرسم

البرنامج شكل مستطيل أو بيضاوى بعيداً عن الشاشة ثم يعرض الصورة الناتجة باستدعاء الـ CopyRect.

فى القائمة (٧-١٣)، تقوم سجلات الـ Source والـ Dest TRect بتحديد حجم bitmap offscreen، وكذلك المكان الذى سيتم فيه نسخ الصورة على الـ form. قم بإنشاء bitmap object باستدعاء الـ TBitmap.Create. عين قيم الـ Width والـ Height لإنشاء bitmap بالحجم الذى تحتاجه.

لرسم الـ offscreen، استدع methods مثل الـ Rectangle والـ Ellipse، وعين قيم لخصائص الـ Pen والـ Brush الاساسية والفرعية. تأكد من استدعاء الـ methods والقيام بالتعينات للـ Canvas الخاص بالـ bitmap. عندما تنتهى من إنشاء صورتك، استدع الـ Canvas method، التابعة للـ PaintBox أو الـ form، مثل الـ CopyRect لإظهار الـ offscreen image.

ان استدعاء الـ TBitmap.Create ينشئ memory device context، والذى يستهلك وقت الـ Windows resources. فى هذا المثال والذى فيه ضغطة زر بسيطة، ستكون النتيجة النهائية سريعة بدرجة كافية، ولكن للحصول على افضل سرعة من خلال loop أو OnPaint، إنشئ bitmap object فى OnPaint event handler الخاص بالـ form ودمره فى الـ OnDestroy.

عند استخدام عبارات الـ with، كن حريصاً على ان ترسم داخل الـ Canvas المستهدفة. لرسم الـ offscreen، استدع الـ Canvas methods الخاص بالـ TBitmap. انسخ الـ offscreen، استدع الـ Canvas methods التابعة للـ PaintBox أو الـ form.

### ،Drag-and-drrp objects

عندما تصمم تطبيق للـ Delphi، فإنك تضغط وتسحب component objects على الـ form. ولكن كيف تستطيع ان تفعل نفس الشئ مع objects الجرافيك فى تطبيقاتك والاجابة تتطلب بعض البرمجة والتى لا يصعب فهمها، ولكنها لا تتطلب انتباه خاص للتفاصيل خاصة فى التعامل مع احداثيات الفأرة.

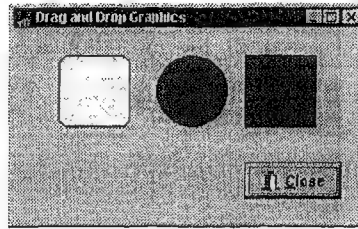
القائمة (١٣-٧) :OnClick الخاص بهذا ال Button  
يوضح كيفية الرسم داخل ال offscreen bitmap

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
  OffScreen: TBitmap;
  Dest, Source: TRect;
begin
  Source := Rect(0, 0, 255, 255);
  Dest := Rect(10, 10, 265, 265);
  OffScreen := TBitmap.Create;
  try
    OffScreen.Width := Source.Right + 1;
    OffScreen.Height := Source.Bottom + 1;
    with OffScreen.Canvas do
      begin
        Pen.Color := clRed;
        Brush.Color := clAppWorkSpace;
        Rectangle(0, 0, 255, 255);
        Ellipse(63, 63, 127, 127);
      end;
    Canvas.CopyRect(Dest, OffScreen.Canvas, Source);
  finally
    OffScreen.Free;
  end;
end;
```

على القرص المدمج، كمثال على ال drag-and-drop objects  
الجرافيكية، جرب تطبيق ال DragMe على القرص المدمج.



توجد الملفات في دليل ال Source\DragMe. كما هو موضح في الشكل  
(١٣-٣)، يعرض البرنامج ثلاثة objects : مستطيل دائري اصفر، دائرة حمراء،  
ومربع أزرق. يعتبر كل منها object هو من ال Shape component object. قم  
بتشغيل البرنامج واضغط واسحب ال objects حول نافذة التطبيق، توضح القائمة  
(١٣-٨) الخطوات اللازمة لإنشاء هذه الخدع والذي يمكنك تشغيلها بمعنى أن أى  
object ترسمه قابل للمسح.



شكل (١٣-٣): يعرض تطبيق الـ DragMe ثلاثة Shapes، التي يمكنك ضغطها وسحبها وتحريكها حول نافذة البرنامج

القائمة (١٣-٨): DragMe\Main.pas

```
unit Main;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Buttons;

type
  TMainForm = class(TForm)
    Shape1: TShape;
    Shape2: TShape;
    Shape3: TShape;
    BitBtn1: TBitBtn;
    procedure ShapeMouseDown(Sender: TObject;
      Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure ShapeMouseMove(Sender: TObject;
      Shift: TShiftState; X, Y: Integer);
    procedure ShapeMouseUp(Sender: TObject;
      Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure FormCreate(Sender: TObject);
  private
    Dragging: Boolean;      // Drag operation in progress
```

```

XOffset, YOffset: Integer; // Offsets from shape upper
left
    FocusRect: TRect;      // Dotted outline while
dragging
    PS: TShape;            // Reference to shape dragging
public
{ Public declarations }
end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.ShapeMouseDown(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    Dragging := True;    { Set dragging flag true }
    XOffset := X;        { Keep offsets from shape upper left }
    YOffset := Y;
    PS := Sender as TShape; { Assign reference to shape }
    with PS do           { Create outline rectangle }
        FocusRect := Rect(Left, Top, Left + Width, Top + Height);
        Canvas.DrawFocusRect(FocusRect); { Draw outline }
end;

procedure TMainForm.ShapeMouseMove(Sender: TObject;
    Shift: TShiftState; X, Y: Integer);
begin
    if Dragging then { Move outline only if dragging }
        begin

```



```

Canvas.DrawFocusRect(FocusRect); { Erase outline }
  with FocusRect do
    begin { Move outline rectangle }
      Left := (PS.Left + X) _ XOffset;
      Top := (PS.Top + Y) _ YOffset;
      Right := PS.Width + Left;
      Bottom := PS.Height + Top;
    end;
    Canvas.DrawFocusRect(FocusRect);
  end;
end;

procedure TMainForm.ShapeMouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Dragging then { Move shape only if dragging }
  begin
    Canvas.DrawFocusRect(FocusRect);
    Dragging := False;
    with Sender as TShape do
      begin { Move shape to new location }
        Left := (Left + X) _ XOffset;
        Top := (Top + Y) _ YOffset;
      end;
    end;
  end;
end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
  Dragging := False;
end;

end.

```

## الباب الثالث عشر : تطوير تطبيقات الجرافيكية

يعرف ال DragMe خمسة متغيرات فى ال module form class . تكون ال Dragging محددة بـ True اثناء عملية السحب والاسقاط . يعتبر ال XOffset وال YOffset هما احداثيات ال pixel لـ cursor الفأرة المرتبطة بالـ Shape الذى يتم سحبه . هذه القيم تجعل من السهل " انتقاء " ال object بضغط الفأرة فى أى مكان داخل حدودها . ال FocusRect هو سجل ال TRect ذا قيم Left ، Top ، و Right ، Bottom التى تحدد المستطيل الخارجى الذى تراه اثناء ضغط وسحب ال object ما . وال PS هى ال pointer للـ objects ، والذى يستخدمها البرنامج للإشارة للـ objects الذى يتم سحبه .

تقوم ثلاثة من ال event handlers بتنفيذ السحب والاسقاط لـ Shape . عندما تضغط ال cursor الفأرة داخل ال Shape ، يبدأ ال procedure ShapeMouseDown عملية السحب والاسقاط أولاً ، يحدد البرنامج ال Dragging flag بـ True . ثم يحفظ قيم احداثيات ال x- وال y- للفأرة فى ال XOffset وال YOffset . هذا يحدد مكان الفأرة بالنسبة للركن الايسر العلوى للـ object . يحفظ البرنامج ايضاً ال Sender parameter ، الذى تم إلقاء الى ال TShape object ، فى ال PS كـ reference فى المستقبل للـ object الذى تم سحبه . باستخدام ال PS ، يعين البرنامج قيم للـ FocusRect لرسم ال outline المنقوت خلال السحب بهذه العبارة :

```
Canvas.DrawFocusRect(FocusRect); { Draw outline }
```

DragFocusRect يرسم ال outline مستطيل منقوت باستخدام منطق ال OR . باستدعاء ال DragFocusRect مرتين لنفس سجل ال TRect يتم مسح ال outline . عند ما تحرك الفأرة ، اذا كان ال Dragging محدد بـ True ، فإن ال OnMouseMove ، التابع للـ ShapeMouseMove ، يستدعى ال DragFocusRect لمسح المستطيل فى موقعة القديم . يعين البرنامج عندئذ قيم جديدة لـ أعضاء ال Left ، ال Top ، ال Right ، وال Bottom التابعين للـ FocusRect . عند قراءة هذا ال code ، تذكر ان ال parameters احداثيات الفأرة x و y مرتبطة بالـ objects الذى يتم سحبه ، وليس بالنافذة أو الشاشة . وبذلك ، فإن موضع ال FocusRect الجديد يساوى مكان ال object (والذى لم يتغير) زائد ال x

و ال y ، ناقص ال offsets من الركن الايسر العلوى للـ object للمكان الذى ضغطت فيه الفأرة . ان استدعاء ال DragFocusRect مع هذه القيم الجديدة ينقل outline عند ما تسحب الفأرة .

عندما تطلق زر الفأرة ، فإن ال event handler الثالث والاخير ShapeMouseUp ، يحرك ال object الى مكانه الجديد . (يجب التأكد من أن ال Dragging محدد بـ True ، فى هذه الحالة ليس ضرورياً أن يسبقه ال OnMouseUp event ، ولكن من الأفضل دائماً أن تكون حذراً) . مرة اخرى ، استدعى البرنامج ال DragFocusRect ، هذه المرة لرسم ال outlined النهائي . بعد تحديد ال Dragging flag بـ False ، فمن السهل ان تحرك ال object بتعيين قيم جديدة لخصائص ال Left ، ال Top له ، مكملأ الخدع الخاصة بالـ object الى الموضوع الأخير المعروف للمستطيل الذى تم تحديده .

تذكر ان ال object يمكن ان تتشارك فى ال event handlers . على سبيل المثال ، لإضافة object جديد قابل للسحب للـ DragMe ، أضف Shape وحدد ال OnMouseDown ، وال OnMouseMove ، وال OnMouseUp له بالـ event handlers الموجودة للـ module .

## Picture Dialogs

يمكنك استخدام dialog components معيارية كالـ OpenFileDialog وال SaveDialog للتصفح أو لحث المستخدمين على اسماء ملفات جرافيكية . ولكن ، هناك components جديدان نسبياً ، وهما ال OpenFileDialog وال SavePictureDialog ، يوفر ان dialogs خاصية لتصفح ملفات الجرافيك . تعرض ال dialogs صور الجرافيك بحيث يستطيع المستخدمون رؤية الملفات التى يختارونها قبل تحميلها فى تطبيق . وتسمى فى بعض الأحيان thumbnail sketch ، بالرغم من اننى لا ارى معنى لهذه التسمية) .

## الـ OpenPictureDialog

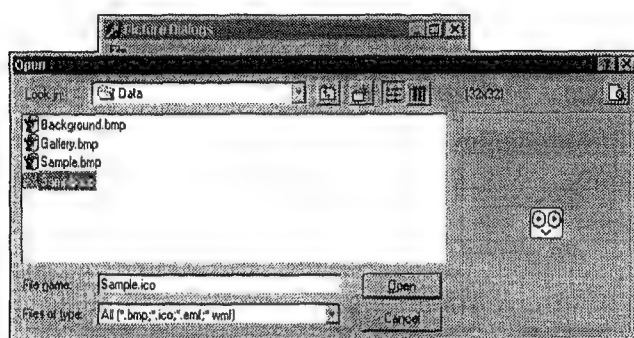
على القرص المدمج: استخدم ال OpenPictureDialog لحث المستخدمين على اختيار اسماء ملفات جرافيك . حسب النظام



## الباب الثالث عشر : تطوير تطبيقات الجرافيكية

الافتراضى ، يوضح ال dialogs كل ملفات ال (bitmap). bmp ، ال (ico). ico (ايقونة) ، ال (enhanced meta file). emf ، وال (meta file). wmf فى الدليل المختار .

عندما يختار المستخدم واحداً من انواع الملفات هذه ، يعرض ال dialog الصورة . [انظر شكل (١٣-٤) . يعرض ال object يوضح ال dialog الشكل الذى تم عرضه بواسطة التطبيق PicDialogs على القرص المدمج فى دليل ال Source\PicDialogs . توضح القائمة (١٣-٩) ال source code للبرنامج .



شكل (١٣-٤): ال OpenPictureDialog يبحث المستخدم  
على اختيار اسماء ملفات الجرافيك، ويعرضها

القائمة (١٣-٩): PicDialogs\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics,  
Controls,
```

```
Forms, Dialogs, Menus, ExtDlgs, ExtCtrls;
```

```
type
```

```
TMainForm = class(TForm)
```

```
Image1: TImage;
```

```
OpenPictureDialog1: TOpenPictureDialog;
```

```
////////////////////////////////////
```

```

SavePictureDialog1: TSavePictureDialog;
MainMenu1: TMainMenu;
File1: TMenuItem;
Open1: TMenuItem;
Save1: TMenuItem;
N1: TMenuItem;
Exit1: TMenuItem;
procedure Open1Click(Sender: TObject);
procedure Exit1Click(Sender: TObject);
procedure Save1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
```

```

var
  MainForm: TMainForm;
```

```
implementation
```

```
{ $R *.DFM }
```

```

procedure TMainForm.Open1Click(Sender: TObject);
begin
  with OpenPictureDialog1 do
    if Execute then
      begin
        Image1.Picture.LoadFromFile(Filename);
        Caption := Lowercase(Filename);
        Save1.Enabled := True; // Enable File|Save... command
      end;
end;
```

```

procedure TMainForm.Save1Click(Sender: TObject);
begin
```

```

with SavePictureDialog1 do
begin
  Filename := Caption;
  if Execute then
  begin
    Image1.Picture.SaveToFile(Filename);
    Caption := Lowercase(Filename);
  end;
end;

end;

procedure TMainForm.Exit1Click(Sender: TObject);
begin
  Close;
end;

end.

```

لاستخدام الـ `OpenPictureDialog` ، أضف الـ `component` على `forme` ثم قم ببرمجة الـ `event handler` مثل الـ `Open1Click` في القائمة والذي يتم تنفيذه عندما يختار المستخدم امر الـ `Open` الخاص ببرنامجك - من قائمة ، مثلاً أو بضغط زر . في هذه النقطة ، استندع امر الـ `Execute` للـ `OpenPictureDialog` ، والذي يرجع بـ `True` فقط اذا اختار المستخدم ملف جرافيك . في اغلب الحالات ، تستخدم `code` مثل التالية :

```

with OpenPictureDialog1 do
if Execute then
begin
  // Use Filename property here to load file
end;

```

ان عبارة الـ `with` تخبر الـ `Delphi` ان يستخدم خصائص و `methods` الـ `OpenPictureDialog` حسب النظام الافتراضى . وعبرة الـ `if` تستدعى الـ `Execute` ، والذي ، اذا كان `True` ، ينفذ العبارات الواقعة بين الـ `begin` والـ `end` . إدخل برمجك هنا لإستخدام خاصية الـ `Filename` ، التى تحمل اسم المسار

للملف المختار. اذا رجعت الـ `Execute` بـ `False`، فإن المستخدم قد ضغط زر الـ `Cancel` أو ضغط `Esc` لإنهاء الـ `dialog`، ولا يجب ان يقوم برنامجك بتحميل ايه ملف.

### خصائص الـ `OpenPictureDialog`:

فيما يلي بعض الخصائص التي يمكنك تعديلها للتعامل مع الـ `OpenPictureDialog` object:

● **DefaultExt**: حدد هذه الخاصية بامتداد اسم الملف الافتراضى، دون نقطة، مثل `bmp` أو `ico`. هذا الـ `string` يتم إلحاقه باسم ملف مختار، الا اذا كان اسم الملف هذا متتبعياً بالفعل بامتداد مسجل. اذا اختار المستخدم ملف له امتداد غير مسجل (على سبيل المثال، يكتب المستخدم `Anyfile` بدلاً من `Anyfile.bmp`)، فإن الـ `DefaultExt` يتم إلحاقها باسم الملف. يمكنك أيضاً تحديد الـ `DefaultExt` فى `code` بامتداد الجرافيك الافتراضى لـ `class` الجرافيك مثل الـ `TBitmap`، باستخدام عبارة مثل:

```
OpenPictureDialog1.DefaultExt := GraphicExtension(TBitmap);
```

● **Files**: اذا كانت خاصية الـ `Options` تتضمن خيار الـ `ofAllowMultiSelect`، إذن تحتوى خاصية الـ `Files` على قائمة `string` بكل الملفات المختارة. والـ `Files.Count` تساوى عدد الملفات المختارة.

● **Files**: هذه تعمل تماماً مثل خاصية الـ `Files` فى `dialogs` فتح الملف (وحفظ الملف) الأخرى. اضغط الزر البيضاوى التالى لهذه الخاصية لتعديل الصفة الافتراضية للـ `file filters`، والتى هى حسب النظام الافتراضى، محددة بالقائمة التالية من الـ `strings` فى الجدول (١٣-١). والتحديدات الافتراضية مسجلة بالـ `TGraphic class`. يمكنك الإضافة إليها باستدعاء الـ `TPicture.RegisterFileFormat`.

● **FilterIndex**: حدد هذه بفهرس الـ `string` فى `string-list` filter التى تريد ان يستخدمها البرنامج حسب النظام الافتراضى. هذا هو نوع اسم الملف الذى يراه المستخدمون عندما يفتحون الـ `dialog` لأول مرة. على سبيل المثال، ان

## الباب الثالث عشر : تطوير تطبيقات الجرافيكية

استخدام الـ filters الافتراضية الموجودة في الجدول السابق ، وتحديد الـ FilterIndex باثنين يؤدي الى اختيار الـ Icons (\*.ico) كاسم افتراضى . اذا كان الـ FilterIndex خارج المدى ، يتم استخدام الـ Filter string الأول حسب النظام الافتراضى .

● **InitialDir** : حدد هذه باسم المسار الدليل الذى تريد ان يراه المستخدمون عندما يفتحون الـ dialog لأول مرة . اذا لم تحدد هذا الحقل ، فإن الدليل الحالى يتم استخدامه حسب النظام الافتراضى .

● **Options** : يمكنك اختيار خيارات متعددة بتعيين قيم لهذه المجموعة . على سبيل المثال ، ان استخدام على سبيل المثال ، الـ AllowMultiSelect لتسمح للمستخدمين باختيار أكثر من اسم ملف واحد (انظر خاصية الـ Files) . سوف اوضح المزيد عن الـ Options فى الفصل التالى .

● **Title** : عين الـ string الذى تريد للـ dialog بأن يعرضه فى الـ title bar للنافذة . اذا لم تعين string لهذه الخاصية ، يكتب الـ dialog كلمة Open على الـ Title . (يعرض الـ SavePictureDialog كلمة Save) .

الجدول (١٣-١) ، خصائص الـ Filter الافتراضية  
للـ OpenPictureDialog والـ SavePictureDialog

Filter name	Filter
All (*.bmp;*.ico;*.emf;*.wmf)	*.bmp;*.ico;*.emf;*.wmf
Bitmaps (*.bmp)	*.bmp
Icons (*.ico)	*.ico
Enhanced Metafiles (*.emf)	*.emf
Metafiles (*.wmf)	*.wmf

### خيارات الـ OpenPictureDialog :

إن الـ OpenPictureDialog والـ SavePictureDialog يقدمان خيارات عديدة يمكنك اختيارها لى تخصص ظهور الـ dialog وعمله . والمجموعة الكاملة للخيارات يتم تعريفها من نوع Pascal enumerated data :



```

TOpenOption = (ofReadOnly, ofOverwritePrompt,
ofHideReadOnly,
ofNoChangeDir, ofShowHelp, ofNoValidate,
ofAllowMultiSelect,
ofExtensionDifferent, ofPathMustExist,
ofFileMustExist,
ofCreatePrompt, ofShareAware, ofNoReadOnlyReturn,
ofNoTestFileCreate, ofNoNetworkButton,
ofNoLongNames,
ofOldStyleDialog, ofNoDereferenceLinks);
TOpenOptions = set of TOpenOption;

```

فى واقع الأمر إنها تعريفان . ال TOpenOption (مفردة) هى النوع ال enumerated - يمكن تعيين أى من الثوابت المذكورة لمتغير من هذا النوع . وال TOpenOption (جمع) يتم تعريفه كثوابت ال TOpenOption لمجموعة Pascal . ويمكن تعيين واحداً أو أكثر أو لا شئ من مجموعات ثوابت ال TOpenOption لمتغير من نوع ال TOpenOption . على سبيل المثال ، لتحديد خيارات ال ofReadOnly وال ofAllowMultiSelect ، يستطيع البرنامج أن ينفذ عبارة مثل :

```

OpenPictureDialog1.Options := [ofReadOnly,
ofAllowMultiSelect];

```

لتعيين خيارات فى وقت التصميم ، اختر ال OpenPictureDialog object واضغط مرتين علامة الزائد الصغيرة الى اليسار من ناحية ال Object فى ال Object Inspector . هذا يفتح القيم الفرعية لتلك الخصائص ، والتي هى مبرمجة كخصائص True/False مقابلة لثوابت ال TOpenOption enumerated . على سبيل المثال ، لتقوم بالتعيين السابق فى وقت التصميم ، حدد خصائص ال ofReadOnly وال ofAllowMultiSelect بـ True .

الجدول (١٣-٢) يصف كل خيار بالترتيب الأبجدي . لا توجد خيارات خاصة بال OpenPictureDialog أو SavePictureDialog الجديد .

الخيار	الوصف
ofAllowMultiSelect	يسمح باختيار ملفاً واحداً أو أكثر . ويتم إدخال اسماء الملفات المختارة فى خاصية الـ Files . ويساوى الـ Files.Count عدد اسماء الملفات المختارة
ofCreatePrompt	إذا أدخل المستخدم اسم ملف غير موجود، يقدم هذا الخيار dialog يسأل عما إذا كان يجب إنشاء ملف جديد بهذا الاسم . يجب أن يقوم الـ code بإنشاء الملف الفعلى - هذا الخيار يبعث التساؤل فقط ، ولكنه لا ينشئ الملف
ofExtensionDifferent	هذا الخيار يعمل كـ flag إما بـ True أو False والذي يشير الى اذا ما كان امتداد اسم الملف مختلف عن الصفة الموجودة فى خاصية الـ DefaultExt . استخدم عبارة مثل : if TOpenOption.ofExtensionDifferent in OpenPictureDialog1.Options then. لتفحص ما اذا كان هذا الـ flag متضمن فى مجموعة الـ Options .
ofFileMustExist	استخدم هذا الخيار لتجعل الـ dialog يعرض رسالة خطأ إذا حاول المستخدم إدخال اسم ملف غير موجود . هذا الخيار يضمن أن خاصية الـ Filename المدخلة تساوى اسم ملف تم اختياره من قائمة الدليل
ofHideReadOnly	إن dialogs الصورة تكون لديها عادة Open As . Read Only check box
ofNoChangeDir	إذا أردت إعادة تحديد الدليل بمساره قبل أن يتم استدعاء عبارة الـ Execute الخاصة بالـ dialog ، ليحتوى هذا

الخيار . اذا لم يحتويه فإن الدليل الحالى يتغير الى أى مسار يختاره المستخدم .	ofNoDereference Links
فى العادى، تكون اختصارات الـ Windows مترجمة الى المسارات الفعلية للملفات التى تشير إليها الاختصارات . اذا لم تكن تريد ان يحدث هذا فتحتوى هذا الخيار . فى هذه الحالة، فإن الـ Filename الذى يتم ادخاله لاختصار اسم مختار سيتم تحديده للملف الـ .lnk الخاص بذلك البند، وان تحديد الملف المشار إليه فعلياً امر يرجع إليك .	ofNoLongNames
يىطل اسماء الملفات الطويلة، ويعرض فقط اسماء النمط القديم ٨,٣ (ثمانية رموز لاسم الملف، ثلاثة رموز الامتداد) فى dialog directory pane .	ofNoNetwork Button
هذا الخيار يحذف زر الـ Network من dialog اختيار الملف، والذى يظهر فقط اذا ما احتواه أيضاً خيار الـ .OldStyleDialog .	ofNoReadOnly Return
اذا حاول المستخدم ان يفتح ملف قراءة فقط، فإن هذا الخيار يجعل الـ dialog يعرض رسالة خطأ ويرفض إدخال الملف المختار . إذ أن استخدام هذا الخيار يضمن ان ملفاً يمكن كتابته ايضاً (ولكن احترس من ان حالة الملف قد تتغير بواسطة برنامج آخر فى الوقت الحالى)	ofNoTestFileCreate
استخدام هذا الخيار يحذر شديد . يهدم حماية ملف الشبكة متيحاً الفرصة لحفظ الملف فى دليل شبكة مشترك فى العادى، يتم فحص اسماء الملف لأى رمز غير مسموح به استخدم هذا الخيار فقط اذا كانت تلك الرموز مستخدمة نوعاً ما فى اسماء الملفات الفعلية	ofNoValidate

يتحول الى الشكل القديم للـ dialog (أنظر أيضاً الـ ofNoNetworkButton)	ofOldStyleDialog
لـ dialog الحفظ فقط ، يعرض هذا الخيار رسالة خطأ تحذر المستخدمين من أنهم قد اختاروا اسم ملف موجود . اذا استخدمت هذا الخيار ، يمكنك ان تتأكد من ان المستخدمين قد أجابوا بـ Yes لهذا النص الـ "Overwrite file?" ، ويمكنك بأمان ان تكتب للملف المختار حتى اذا كان موجوداً بالفعل	ofOverwritePrompt
اذا ادخل المستخدم مسار دليل غير موجود ، هذا الخيار يعرض رسالة خطأ ويرفض إدخال المسار في الـ Filename	ofPathMustExist
يحدد أن الملفات المحددة بصفة القراءة فقط التي يمكن إدخالها بواسطة الـ dialog في خاصية الـ Filename	ofReadOnly
استخدمه يحذر شديد . هذا الخيار يهدم انتهاكات التشارك في ملف (المستخدمون يحاولون ان يفتحوا ملفاً يتم استخدامه ، مثلاً) ، ويسمح بالاختيار من ملفات قد تكون في اثناء عملية تعديل من قبل مستخدمين أو برامج اخرى	ofShareAware
استخدم هذا الخيار لعرض زر الـ Help . وكذلك قم بتعيين قيمة الـ HelpContext لعرض ملف الـ help الخاص بك	ofShowHelp

### الـ SavePictureDialog

على القرص المدمج: ان الـ SavePictureDialog هو تقريرياً الـ  
OpenPictureDialog . وكلا الـ components لهما خصائص  
وخيارات متطابقة ، ولكن بالطبع ، ان الـ SavePictureDialog يحث  
المستخدمين على اسماء ملفات لحفظ ملفات الجرافيك . إن برنامج العرض



PicDialogs الموجود على القرص المدمج فى دليل ال Source\PicDialogs يظهر كيفية استخدام ال SavePictureDialog [راجع شكل (١٣-٤)] والقائمة (١٣-٩).

انك تستخدم ال SavePictureDialog بنفس الطريقة تقريباً التى تستخدم بها ال OpenFileDialog. ولكن، قد تريد ان تأخذ خطوة واحدة إضافية كما هو موضح فى ال code التالية التى توضح كيفية الاستجابة لزر أو أمر ال Save الخاص بالبرنامج:

```
With SavePictureDialog1 do
begin
  Filename := Caption;
  if Execute then
  begin
    Image1.Picture.SaveToFile(Filename);
    Caption := Lowercase(Filename);
  end;
end;
```

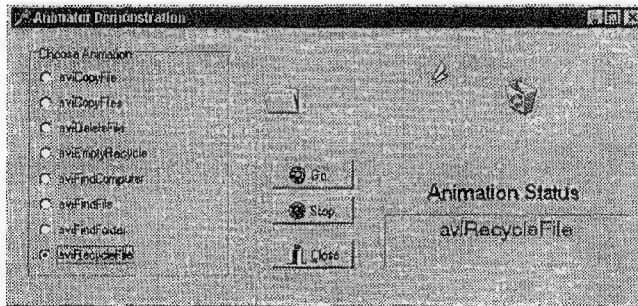
هذه ال code، المأخوذة من التطبيق PicDialogs، تستخدم عبارة with لتحديد ان تلك الخصائص وال methods التى تنتمى لل SavePictureDialog1 يجب ان يتم استخدامها. ان خاصية ال Filename لذلك ال object يتم تحديدها أولاً فى ال window caption- هذا يجعل ال dialog يعرض اسم الملف الحالى، الذى تم حفظه فى ال caption النافذة بواسطة ال close-command procedure (انظر ال Open1Click فى القائمة). ان استدعاء ال Execute يظهر ال dialog، ويدخل True فقط اذا اختار المستخدم اسم ملف وضغط زر ال Save. فى هذه الحالة، فان ال SaveToFile method الخاص بال Image فى خاصية ال Picture يتم استدعاءه لكتابة الملف على القرص. ولان المستخدم قد يكون اختار اسم ملف مختلف ليحفظ فيه الملف، فان ال caption النافذة يتم إعادة تحديده بخاصية ال Filename الحالية لل SavePictureDialog1. هذا يضمن ان اسم الملف المختار و ال caption النافذة متفقان دائماً.

## :Animations

مع ال Delphi ، فإنه من السهل إضافة صور متحركة لبرنامجك . في هذا الفصل ، سوف اشرح اسس استخدام ال Animate وال MediaPlayer . مع ال Animate ، يمكنك إضافة صور متحركة معيارية تعطى تغذية خلفية بصرية للمستخدمين اثناء عمليات عامة مثل نسخ وحذف الملفات . يمكنك ايضاً ان تلعب بأى ملف avi . (سمعى بصرى) . مع ال MediaPlayer ، يمكنك إنشاء متصفح ملف avi . ذى خصائص كاملة ، كما يوضح تطبيق ال VideoPlayer الخاص بهذا الفصل .

### : Animate component

**على القرص المدمج:** ان ال Animate على ال Win32 palette يمكن ان يدير أى ملف AVI ليوفر صور جرافيكية متحركة فى النافذة . يمكنك ايضاً استخدام هذا ال component لعرض قصاصات ال AVI العامة المتوفرة مع ال Windows كذلك المعروضة بواسطة ال Explorer اثناء عمليات الملف . يوضح شكل (١٣-٥) واحدة من ثمانى صور متحركة معيارية ممكنة معروضة بواسطة التطبيق Animator الموجود على القرص المدمج فى دليل ال Source\Animator . قم بتحميل ملف مشروع هذا البرنامج ، Animator.dpr ، فى Delphi ، واضغط F9 لتشغيل برنامج العرض . اختر واحداً من ازرار ال radio لترى صورة متحركة مختلفة . اضغط زر ال Go لتشغيل الصور المتحركة ؛ اضغط Stop للتوقف . توضح القائمة (١٣-١٠) ال source code لبرنامج ال Animator .



شكل (١٣-٥): التطبيق Animator يوضح كيفية استخدام ال Animator لتشغيل AVI معيارية لـ Windows ، فى هذه الحالة ، هى صورة ال "Recycle File"

Animator\Main.pas : (۱۰-۱۳) القائمة

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls,
  Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, ComCtrls;

type
  TMainForm = class(TForm)
    Animate1: TAnimate;
    RadioGroup1: TRadioGroup;
    GoBitBtn: TBitBtn;
    StopBitBtn: TBitBtn;
    BitBtn1: TBitBtn;
    StatusText: TStaticText;
    Label1: TLabel;
    procedure GoBitBtnClick(Sender: TObject);
    procedure StopBitBtnClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

{ Define array types for easier assignments of the
  Animate object's CommonAvi property, and also the
  StatusText object that shows which animation is
  running. }
type
```

```
aviKindArray = array[0 .. 7] of TCommonAvi;
aviStringArray = array[0 .. 7] of String;

{ Define constant arrays containing the TCommonAvi
values
in the same order they appear in the RadioGroup object,
and also strings for displaying in the StatusText object
that shows which animation is running. }
const

aviKinds: aviKindArray =
  (aviCopyFile,
   aviCopyFiles,
   aviDeleteFile,
   aviEmptyRecycle,
   aviFindComputer,
   aviFindFile,
   aviFindFolder,
   aviRecycleFile);

aviStrings: aviStringArray =
  ('aviCopyFile',
   'aviCopyFiles',
   'aviDeleteFile',
   'aviEmptyRecycle',
   'aviFindComputer',
   'aviFindFile',
   'aviFindFolder',
   'aviRecycleFile');

{ Start selected animation. This event handler is assigned
to the OnClick event for both the Go button and the
RadioGroup1. Clicking Go or clicking a radio button
starts
the animation immediately. }
procedure TMainForm.GoBitBtnClick(Sender: TObject);
var
  AnimIndex: Integer; // Index of selected animation
begin
```



```

AnimIndex := RadioGroup1.ItemIndex;
with Animate1 do
begin
    StatusText.Caption := aviStrings[AnimIndex];
    CommonAVI := aviKinds[AnimIndex];
    Play(1, FrameCount, 0); // Start the animation
end;
end;

{ Halt the animation when user clicks the Stop button. }
procedure TMainForm.StopBitBtnClick(Sender: TObject);
begin
    Animate1.Stop;
    StatusText.Caption := '(stopped)';
end;

end.

```

يوضح تطبيق الـ Animator كيفية اختيار، بدء، وإيقاف الـ Windows AVClip المعيارية باستخدام الـ Animator. ان الـ GoBitBtnClick procedure يتم تعيينه للـ OnClick لكلاً من زر الـ Go والـ RadioGroup1 وكنتيجة لذلك، فان ضغط الـ Go أو اختيار صورة متحركة من نوع زر الـ radio يؤدي الى بدء تشغيل الصورة المتحركة على الفور.

يبدأ الـ GoBitBtnClick procedure بتعيين الـ ItemIndex الخاص بالـ RadioGroup1 للـ Integer المحلي، حتى يجعل استخدامه اسهل. هذا يشير الى زر المجموعة الذي يتم اختياره حالياً. لتظهر على الشاشة أى صور متحركة يتم تشغيلها، يتم تعيين string للـ StaticText object من constant array، aviStrings معرفة فى قطاع الـ implementation الخاص بالـ unit. باستخدام نفس فهرس مجموعة الـ radio، فإن خاصية الـ CommonAvi للـ Animate يتم تحديدها باسم الصورة المتحركة المختارة مثل aviCopyFile أو aviFindComputer. لإدارة الصورة المتحركة، يستدعى البرنامج الـ Play method الخاص بالـ Animate باستخدام العبارة:

Play(1, FrameCount, 0);

ان arguments الأولتين تشيران الى مدى اعداد الاطار الذى يتم إدارتها . فالاطار الاول يحمل الرقم واحد . وال FrameCount ، وهو حقل فى ال Animate ، يشير الى عدد الإطارات فى ال argument AVI Clips . الأخيرة تساوى عدد التكرارات . حدد هذه القيمة بصفر ، كما هو الحال هنا ، لتعيد عرض الصورة المتحركة حتى يتم استدعاء ال Stop method الخاص بال Animate .

هذا يحدث فى ال Animate عندما تضغط زر ال Stop . هذا يؤدي الى استدعاء StopBtnClick ، الذى يستدعى ال Stop method الخاص بال Animate . للإشارة الى انه لا يوجد تشغيل لصورة متحركة ، يحدد البرنامج ايضاً ال Caption الخاص بال StatusText بقراءة '(stopped)' .

### •Animate properties

فيما يلي خصائص ال Animate والتي يمكنك استخدامها لتخصيص ال Animate :

● **Active** : حدد هذه بـ True لتبدأ تحريك الصورة المتحركة . ولكن قبل تحديد ال Active بـ True يجب ان تختار AVI Clip معيارية بخاصية ال CommonAVI ، أو تدخل اسم ملف لصورة ال AVI فى حقل ال FileName . ان تحديد ال CommonAVI بـ aviNone ، أو مسح حقل ال FileName ، يحدد ال Active بـ File . يمكنك ايضاً فحص ال Active فى وقت تشغيل لتحديد ما اذا كانت الصورة المتحركة فى حالة تشغيل .

● **AutoSize** : حدد هذه عادة بـ True حتى يتم تحديد حجم ال Animate بصورة تلقائية اعتماداً على حجم ال AVI Clip . حدد ال AutoSize بـ File فقط اذا كانت كل القصصات لها نفس الحجم ، أو اذا كنت تدير Clip واحدة فقط . فكرة : لتحديد كبر حجم ال Animate الذى تقوم به ، حدد ال AutoSize بـ True واختر CommonAVI أو FileName . ثم حدد ال Active و ال AutoSize بـ File . ان ال Animate محدد الآن بالحجم المطلوب ال AVI Clip المختارة .

● **Center** : حدد هذه عادة بـ True بحيث يصبح الصورة المتحركة فى منتصف مسافة عرض ال Animate . تحديد ال Center بـ File تجعل الصورة المتحركة تظهر فى الركن الايسر العلوى من ال Animate .

● **CommonAVI**: هذه الخاصية تختار الـ Windows AVI Clip المعيارية .  
حدد الـ CommonAVI بواحد من الثوابت المعدودة لـ TCommonAvi وهي  
aviFindComputer ، aviFindFile ، aviFindFolder ، aviNone ،  
aviEmptyRecycle ، aviRecycleFile ، aviCopyFile ، aviCopyFiles ، و  
aviDeleteFile .

● **FileName**: ادخل اسم ملف من نوع ملف الـ AVI Clip - فقط (لا صوت) في هذه الخاصية . يمكنك ان تفعل هذا في وقت التصميم أو في وقت التشغيل . هذا يحدد بصورة تلقائية الـ CommonAVI بـ aviNone - لا يمكنك اختيار الـ Windows AVI Clip العامة وملفاً في نفس الوقت؛ يمكنك ان تستخدم إحدى الخاصيتين أو الأخرى . يجب ان يوجد الملف ، ويجب ان يكون الـ AVI Clip صامتة . (انظر الـ "MediaPlayer component" في هذا الباب لمزيد من المعلومات عن كيفية ادارة افلام و الـ Sound Clip) .

● **FrameCount**: يساوي عدد الإطارات في الـ AVI Clip . هذه القيمة هي قراءة فقط ، فهي لا تظهر في الـ Object Inspector . يتم تحديث الـ FrameCount بصورة تلقائية عند تحديد قيم الـ CommonAVI أو الـ FileName .

● **Repetitions**: تحدد بعدد التكرارات المخصصة لإدارة الـ Clip عندما تكون الـ Active بـ True . وقيمة الصفر تشير الى ان Clips يجب ان تدار بلا توقف حتى يتم تحديد الـ Active بـ File أو يتم استدعاء الـ Stop method للـ Animate .

● **StartFrame**: تحدد الإطار البادئ لتبدأ الصورة المتحركة . الإطار الأول يحمل الرقم واحد . إنك لا تحتاج ان تحدد هذا الحقل اذا كان برنامجك يستدعي الـ Play method .

● **StopFrame**: عدد الإطار الأخير الذي يتم إدارته في الصورة المتحركة . يوجد عدد الإطارات في حقل الـ FrameCount قراءة-فقط . إنك لا تحتاج ان تحدد هذا الحقل اذا كان برنامجك يستدعي الـ Play method .

## الباب الثالث عشر : تطوير تطبيقات الجرافيكية

● **Timers** : عندما يحدد بـ false (وهو التحديد الافتراضي)، تؤدي هذه الخاصية بالصورة المتحركة الى التشغيل في thread منفصل . عند تحديده بـ True، تؤدي هذه الخاصية بالصورة المتحركة الى ان تكون محكومة بالنظام العادي، يمكن ان تترك الـ Timers محددة بـ false، ولكن اذا اردت ان تتزامن events اخرى مع الصورة المتحركة، يمكنك تغيير الـ Timers بـ True. استخدم الـ events OnStart والـ OnStop الخاصة بالـ Animate للتحكم في نشاط التزامن.

● **Transparent** : حده بـ True لعرض لون الـ object الأم كخلفية للصورة المتحركة . هذه هي القيمة العادية عند عرض الصورة المتحركة . اذا كانت الـ Transparent محددة بـ false، يؤخذ لون خلفية الصورة المتحركة من الـ AVI Clip.

### الـ MediaPlayer component :

على القرص المدمج: ان الـ MediaPlayer يعتبر استديو منزلي فعلى يمكنك استخدامه لإنشاء برمجيات الوسائط المتعددة . لقد استخدمت الـ MediaPlayer لإنشاء برنامج العرض VideoPlayer على القرص المدمج (في دليل الـ Source\VideoPlayer). وهذا البرنامج يستطيع ان يفتح ويدير أى AVI Sound Clip، ويوضح كيفية إنشاء ميزات مثل خيار التكرار الآلى.

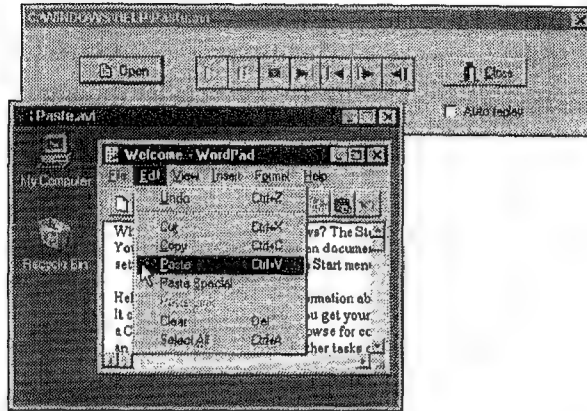


يوضح شكل (١٣-٦) برنامج الـ VideoPlayer وهو يدير AVI Clip ووجدتها في دليل الـ Windows\Help- هناك العديد من الـ Clips مخزنة في هذا المكان وتعتبر جزءاً من نظام الـ online الخاصة بالـ Windows. والـ Clip الموضحة هنا هي صورة توضح القص واللصق. توضح القائمة (١٣-١١) الـ source code للـ VideoPlayer. ان بعض خطوات البرنامج لها تأثير خطير على نتائج وقت التشغيل، لذا اضيفت العديد من التعليقات في القائمة لاوضح العبارات . سوف اشرح المزيد عن عمل البرنامج بعد القائمة.

**ملحوظة:** ان الـ VideoPlayer هو أكثر من مجرد محرك صورة. فهو يمكن ان يعمل كـ controls للوسائط المتعددة للقرص المدمج، أو الـ

Note

MIDI ، أو الـ VCR . فى هذا الباب ، سوف اشرح كيفية استخدام الـ MediaPlayer لعرض AVI Clip صامتة وصوتية ، ولكن نفس الـ component قادر على العمل كـ control للعديد من انواع الوسائط المتعددة . يستطيع الـ MediaPlayer لايضاً ان يدير افلام الـ MPEG اذا قام المستخدم بتركيب برنامج الـ ActiveMovie الخاص بالـ Microsoft .



شكل (١٣-٦) ، تطبيق الـ VideoPlayer على القرص المدمج يمكن ان يعرض أى AVI Sound Picture Clip . والـ Clip الموضحة هنا هى Paste.avi من دليل الـ Windows\Help

#### القائمة (١٣-١١) : VideoPlayer.Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics,  
Controls,
```

```
Forms, Dialogs, StdCtrls, Buttons, MPlayer, ComCtrls;
```

```
type
```

```
TForm1 = class(TForm)
```

```
MediaPlayer1: TMediaPlayer;
```

```
OpenBitBtn: TBitBtn;
```



```

BitBtn2: TBitBtn;
  OpenFileDialog1: TOpenDialog;
  AutoPlayCheckBox: TCheckBox;
  procedure OpenBitBtnClick(Sender: TObject);
  procedure MediaPlayer1Notify(Sender: TObject);
  procedure MediaPlayer1Click(Sender: TObject;
    Bulton: TMPBtnType; var DoDefault: Boolean);
  procedure AutoPlayCheckBoxClick(Sender:
    TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

(* Note: Set the MediaPlayer1 object properties as follows
  AutoEnable    False  All buttons always enabled
  AutoOpen      True   Doesn't really matter
  AutoRewind    True   Rewinds media when it
  stops
  Other properties Default settings
*)
{ Responds to user selection of the Open file button. }
{ Opens media file and starts playing immediately. }
{ Also sets the window caption to the filename. }
procedure TForm1.OpenBitBtnClick(Sender: TObject);
begin
  if OpenFileDialog1.Execute then
    begin

```

```

Form1.Caption := OpenFileDialog1.FileName;
MediaPlayer1.FileName :=
OpenDialog1.FileName;
MediaPlayer1.Notify := True; // Wants media
event notify
MediaPlayer1.Open;          // Opens assigned file
MediaPlayer1.Frames := 1;   // Sets single step
frames
MediaPlayer1.Play;          // Start playing the file
end;
end;

{ Responds to media event notifications. This gives us the
  chance to check whether the auto-replay checkbox is
  set, and
  if so, and if the media is stopped, to restart play.
  Despite
  the control's documentation, it is necessary to check
  whether
  Notify is true. Even if this flag is false, the procedure is
  called when the user clicks the stop button. }
procedure TForm1.MediaPlayer1Notify(Sender: TObject);
begin
  if (MediaPlayer1.Notify) and          // If flag is true
    (MediaPlayer1.Mode = mpStopped) and // and playis
    stopped
    (AutoPlayCheckBox.Checked) then    // &
    checkbox enabled
    begin
      MediaPlayer1.Rewind;              // rewind to start
      MediaPlayer1.Play;                // and begin
playing
    end;
    { You must set Notify to True so that the next media
    event
    generates a notification; otherwise, this procedure
    would be called only once. }
    MediaPlayer1.Notify := True; // Request next
notification

```

end;

{ Responds to user clicking in the MediaPlayer object. We need to check for this because, if the auto-replay checkbox is enabled, stopping the media would generate a notification, which would start playing again! In other words, this procedure allows the media to be stopped regardless of the auto-replay checkbox setting. }

```
procedure TForm1.MediaPlayer1Click(Sender: TObject;
  Button: TMPBtnType; var DoDefault: Boolean);
begin
  if (Button = btStop) or (Button = btPause) then
    MediaPlayer1.Notify := False // Do not continue
    replay
  else
    MediaPlayer1.Notify := True; // Replay if checkbox
    is set
end;
```

{ Responds to user changing the state of the auto-replay checkbox. If the checkbox is being enabled, this procedure turns on notifications so that, when the media stops, the Notify event handler can restart the media playing. }

```
procedure TForm1.AutoPlayCheckBoxClick(Sender:
  TObject);
begin
  if AutoPlayCheckBox.Checked then
    MediaPlayer1.Notify := True; // Triggers notifications
end;

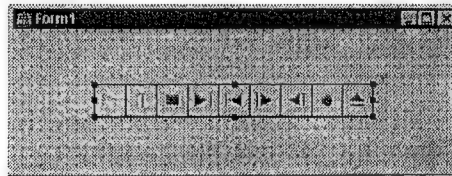
end.
```

هناك العديد من العوامل تتضامن في إنجاح كتابة محرك صورة AVI مثل التطبيق VideoPlayer المذكور هنا . ان نسخة نظام التشغيل ، والآت العرض ،



ومحركات الصورة على نظام المستخدم، و format ملف ال AVI كل هذه العوامل تؤثر على الناتج. ان نظم الحاسوب الحديثة تعطى افضل النتائج- اما تلك التى لها معدات قديمة ونسخ أولية من ال Windows 95 قد تعمل بشكل بطيء أو لا تحرك ال AVI Clip أصلاً. على ايه حال، ان ال MediaPlayer يبدو قوياً بدرجة عالية- فقد استخدمت ال VideoPlayer لتشغيل بعض ال AVI Clips التالفة والتى لم أكن أراها بأية وسيلة أخرى.

على الشاشة، يعتبر ال MediaPlayer مجرد toolbar من ازرار يمكن للمستخدمين ان يتعاملوا معها لتشغيل ال object، مثل الازرار الموجودة على ال VCR. يوضح شكل (٧-١٣) MediaPlayer غير معدل تم إضافته على form. والازرار التسعة بالترتيب من اليسار الى اليمين هى: Play، Pause، Stop، Next، Previous، Step، Back، Record، و Eject. وهناك ثلاث خصائص تؤثر على مظهر الزر. استخدم خاصية ال ColoredButtons لعرض ازرار منفردة بالالوان أو الابيض والاسود. استخدم ال EnabledButtons لتشغيل أو إبطال ازرار منفردة (تعرض الازرار المبطله باللون الرمادى المعتم، ولا يمكن اختيارها). استخدم خاصية ال VisibleButtons للإشارة الى الازرار التى يجب عرضها. هذه مفيدة فى إخفاء الازرار الغير مرغوب فيها- يقوم ال VideoPlayer، مثلاً بإخفاء ازرار ال Record وال Eject التى ليس لها استخدام فى البرامج.



شكل (٧-١٣): MediaPlayer غير معدل بكل ازراره

لبدء إدارة ال AVI Clip، استخدم ال dialog box أو وسيلة أخرى لحث المستخدمين على إعطاء اسم ملف. يعرض ال VideoPlayer كيفية برمجة هذا الفعل فى OpenBitBtnClick، والذى يتم تنفيذه عندما تضغط زر ال Open:   
if OpenFileDialog1.Execute then

```
begin
Form1.Caption := OpenFileDialog.FileName;
MediaPlayer1.FileName := OpenFileDialog.FileName;
MediaPlayer1.Notify := True; // Wants media event
notify
MediaPlayer1.Open;           // Opens assigned file
MediaPlayer1.Frames := 1;    // Sets single step frames
MediaPlayer1.Play;           // Start playing the file
end;
```

ان استدعاء الـ Execute يؤدي الى إظهار نافذة الـ OpenFileDialog .  
 الـ function ترجع بـ True اذا كان المستخدم قد اختار اسم ملف ، في هذه الحالة  
 يتم تنفيذ العبارات الواقعة بين الـ begin و end . لعرض الملف الذي يتم تشغيله ،  
 تحدد العبارة الأولى الـ Caption الـ from باسم الملف المختار ، المأخوذ من الـ  
 OpenFileDialog . نفس هذا الاسم يتم تعيينه لخاصية الـ FileName التابعة للـ  
 MediaPlayer1 - ربما يكون هذا هو الملف الذي يتم تصفحه . للإشارة الى اننا نريد  
 اختطارات عن events الوسائط ، يحدد البرنامج الـ Notify بـ True (المزيد عن هذه  
 النقطة سيأتي فيما بعد) .

ثم ، يقوم الـ Open method بتحميل الملف في الذاكرة ، ولكن لا يقوم الآن  
 بإدارته . للإشارة الى كم إطاراً يجب ان يقوم الزر بتقديمها للصورة ، يتم تحديد الـ  
 Frames بواحد (يتم تحديد هذه عادة بـ ١٠٪ من عدد الاطارات في الصورة ، ولكنني  
 افضل زيادة إطاراً واحداً في كل مرة) . جدير بالذكر ان خاصية الـ Frames غير منشورة -  
 فيمكنك ان تحدها في وقت التشغيل ، ولكن ليس في الـ Object Inspector .

لبدء إدارة الـ AVI Clip ، يقوم باستدعاء الـ Play method للـ  
 MediaPlayer1 . على عكس الـ Animate ، هذه النسخة من الـ Play ليس لها  
 arguments . يتم استدعائها لتبدأ تشغيل الصورة . اذا لم يتم البرنامج باستدعاء  
 الـ Play ، يجب ان يضغط المستخدمون زر الـ Play لبدء تشغيل الـ AVI Clip .

بتحديد خاصية الـ Notify بـ True ، والتي يتم توليدها لـ events متنوعة  
 تضع نقطة بدء وإيقاف الـ Clip ، حدد الـ Notify بـ True (يجب ان تفعل هذا

باستخدام عبارة برنامج كتلك الموضحة في الجزئية السابقة)، وقم بإنشاء  
OnNotify للMediaPlayer. و OnNotify الخاص بالVideoPlayer هو :

```
procedure TForm1.MediaPlayer1Notify(Sender: TObject);
begin
    if (MediaPlayer1.Notify) and           // If flag is true
        (MediaPlayer1.Mode = mpStopped) and // and play's
        stopped
        (AutoPlayCheckBox.Checked) then    // &
        checkbox enabled
    begin
        MediaPlayer1.Rewind;                // rewind to start
        MediaPlayer1.Play;                  // and begin
        playing
    end;
    MediaPlayer1.Notify := True; // Request next
    notification
end;
```

ان هدف هذا ال code هو إعادة إدارة ال AVI Clip الحالية اذا كان  
المستخدم قد اختار check box اختيار إعادة الإدارة Auto. ولان الاخطارات  
المتعددة يتلقاها نفس هذا ال procedure، فإن الخطوة الأولى هي لتحديد ما اذا  
كانت الصورة قد توقفت، في هذه الحالة يتم إرجاعها وإعادة بدئها من البداية. اذا  
لم يتم اختيار AutoPlayCheckBox، فلا يفعل ال procedure شيئاً، ولذلك،  
تتوقف الصورة عند نهايتها، (أو عندما يضغط المستخدم زر ال Stop).

بالرغم من ان البرنامج يحدد ال Notify بـ True، فمزال ضرورياً ان تختار  
ال Notify عند الإدخال للOnNotify. تشير الاختبارات الى ان الاخطارات  
مزال يتم تلقيها في بعض الاحيان عندما تكون ال Notify بـ False، ربما لان تلك  
ال event يتم توليدها في الاوقات التي تقوم فيها ال Notify بتغير القيمة. اذا كانت  
ال Notify محددة بـ True، فإن OnNotify الخاص بالVideoPlayer يحدد  
خاصية ال Mode ليحدد حالة ال object. اذا كانت هذه الحالة تساوى  
mpStopped، إذن يكون المستخدم قد أوقف الإدارة (أو انها قد إنتهت). اذا كان  
ال AutoPlayCheckBox مختاراً، يقوم البرنامج بإرجاع وإعادة إدارة ال Clip

## الباب الثالث عشر : تطوير تطبيقات الجرافيكية

الحالية باستدعاء الـ Play method والـ Rewind الخاصة بالـ MediaPlayer's.

**ملحوظة:** لاحظ ان الـ Notify تكون محددة بـ True قبل انتهاء الـ OnNotify مباشرة. هذا ضرورياً لطلب مزيد من الاخطارات. اذا لم تكن تريد المزيد من الاخطارات، حدد الـ Notify بـ False قبل إدخال الـ OnNotify event.

ونصيحة اخيرة بالنسبة للإخطار هي انه عندما يقوم المستخدم بإيقاف الصورة، يجب ان يتوقف الفعل فوراً بدلاً من إعادة البدء من جديد. كما هو مكتوب، يكرر الـ OnNotify الصورة الحالية اذا كان الـ Auto replay مختاراً- هذا قد يؤدي بقاصة الـ AVI Clip للتكرار حتى اذا تم ايقافها بضغط زر الـ Stop الخاص بالـ controller.

حل هذه المشكلة، يقوم الـ OnClick خاص بالـ MediaPlayer بفحص ما اذا كان المستخدم قد ضغط زر الـ Stop. ها هو الـ procedure الكامل من تطبيق الـ VideoPlaye:

```
procedure TForm1.MediaPlayer1Click(Sender: TObject;
  Button: TMPBtnType; var DoDefault: Boolean);
begin
  if (Button = btStop) or (Button = btPause) then
    MediaPlayer1.Notify := False // Do not continue
    replay
  else
    MediaPlayer1.Notify := True; // Replay if checkbox
    is set
end;
```

يتلقى الـ procedure ثلاثة parameters الـ Sender يعتبر الـ object الذي تم ضغطه (في هذه الحالة هو الـ MediaPlayer1). ولان هناك object واحداً فقط من هذا النوع، فيمكننا ان نتجاهل هذا الـ parameter ونحن مطمئن. يشير الـ Button parameter الى الزر الذي تم ضغطه. وهذا الـ parameter يساوي احدى هذه القيم: btBack ، btEjec ، btNext ، btPause ، btPlay ،

ما اذا كان المستخدم قد ضغط ازرار الـ Stop أو الـ Pause . اذا كان كذلك، يتم تحديد الـ Notify بـ False بحيث لا يستمر تلقى الاخطارات بواسطة الـ OnNotify الخاص بالـ MediaPlayer ، وبذلك، يتم ابطال الـ auto-replay بغض النظر عن قيمة الـ check box .

والـ parameter الأخير ، الـ DoDefault ، الذى يتلقاه الـ OnClick الخاص بالـ MediaPlayer يشير اذا ما كان الـ code أو الـ component يجب ان ينفذ اعمالاً معيارية . اذا كان الـ DoDefault محدد بـ True (وهو الافتراضى) ينفذ الـ code لكل الازرار . ان ضغط الـ Play ، مثلاً، يجعل الـ MediaPlayer يستدعى الـ Play method الخاص به .

اذا لم تكن تريد للـ component ان يستدعى الـ methods . الخاصة به ، حدد الـ DoDefault بـ False فى الـ OnClick التابع للـ MediaPlayer . هذا ينبه الـ component الى ان الـ code قد نفذت عبارات بديلة لزر أو آخر . على سبيل المثال ، لتنتقل من بين اطارات الـ AVI Clip ، قد تستخدم الـ code التالية فى الـ OnClick :

```
begin
  if Button = btStep then
    begin
      MediaPlayer1.Frames := 1;
      MediaPlayer1.Step;
      DoDefault := False;
    end;
    // Other buttons execute standard methods
  end;
```

اذا كان الـ Button parameter يشير الى ان زر الـ Step قد تم ضغطه ، يحدد البرنامج الـ Frames بواحد ويستدعى منهج الـ Step للـ MediaPlayer . هذا يجعل الـ Clip تتقدم باطار واحد بدلاً من ١٠٪ من العدد الكلى للإطارات ، وهو التحديد المعيارى . لتنبيه الـ MediaPlayer الى ان البرنامج قد استجاب بالفعل لزر الـ Step ، فإن الـ DoDefault يتم تحديده بـ False . ولان الـ DoDefault محدد بـ

True حسب النظام الافتراضى ، فبعد ان ينتهى الـ OnClick ، يستجيب الـ MediaPlayer لضغطات زر آخر بشكل عادى .

### خصائص الـ MediaPlayer:

فيما يلى بعض الملاحظات حول الخصائص المختارة للـ MediaPlayer التى يمكنك استخدامها الكى تجعل الـ component object متفق مع رغباتك :

● **AutoEnable** : حدد بـ True لتشغيل وابطال ازرار الـ MediaPlayer المختارة اعتماداً على حالة جهاز الوسائط المتعددة . على سبيل المثال ، عندما تحدد هذه الخاصية بـ True ، يتم ابطال زر الـ Play عندما لا يتم تحميل ملف . حدد الـ AutoEnable بـ False لتشغيل كل الازرار طوال الوقت . ان الـ VideoPlayer يستخدم هذا التحديد .

● **AutoOpen** : حددها بـ True لفتح ملف بصورة تلقائية تم تحديده من جانب الـ FileName . اذا كانت محددة بـ False ، يجب ان تستدعى الـ Open method لفتح الملف .

● **AutoRewind** : حددها بـ True لإرجاع ملف الوسائط المتعددة عندما ينتهى . ان ضغط زر الـ Play عندئذ يعيد بدء الوسائط من بدايتها .

● **DeviceType** : استخدم هذه الخاصية لتحديد أى نوع من الجهاز أو الملف يتحكم فيه الـ MediaPlayer . ان القيمة الافتراضية ، وهى dtAutoSelect ، تحدد نوع الجهاز ، وبذلك نوع محرك الجهاز الذى يستخدم ، على اساس امتداد اسم الملف الذى تم اختياره كما هو مسجل فى سجل الـ Windows . لتحديد نوع معين من الاجهزة ، يمكنك ان تحدد الـ DeviceType باحدى القيم التالية : dtAVIVideo ، dtCDAudio ، dtDAT ، dtDigitalVideo ، dtMMMMovie ، dtOther ، dtOverlay ، dtScanner ، dtSequencer ، dtVCR ، dtVideodisc ، أو dtWaveAudio .

● **Display** : تعين اسم component مثل الـ Panel أو الـ Form لهذه الخاصية . ثم يتم ارسال المخرجات من الـ MediaPlayer لهذا الـ component . ولكن بعض انواع الوسائط تعرض دائماً فى نافذة منفصلة . كنتيجة لذلك ، يتم

تجاهل خاصية الـ Display وسائط الـ Animation ، الـ AVI Video (ذات صوت)، الـ Digital Video ، الـ Overlay ، والـ VCR .

● **FileName**: يعين اسم ملف الوسائط الذى يجب فتحه . اذا كانت الـ AutoOpen محددة بـ True ، فإن التغييرات التى تحدث للـ FileName تؤدي الى استدعاء الـ Open method بصورة تلقائية .

● **Frames**: هذه الخاصية تحدد عدد الاطارات التى تم زيادتها فى الوسائط عندما ضغط المستخدم زر الـ Step أو الـ Back . فى العادى تكون الـ Frames محددة ، بعد فتح ملف الوسائط ، بـ ١٠٪ من عدد الاطارات الموجودة فى الملف . قم بتغيير هذه القيمة بعد استدعاء الـ Open لتغيير الاطارات الزائدة .

● **Mode**: إنك تختبر هذه الخاصية بشكل نموذجى فى الـ OnNotify التابع للـ MediaPlayer (تأكد من ان تحدد الـ Notify بـ True حتى يتلقى الـ procedure اخطارات الوسائط) . ان الـ Mode تساوى واحدة من القيم الواضحة الـ mpNotReady ، mpStopped ، mpPlaying ، mpPaused ، mpSeeking ، mpRecording ، أو الـ mpOpen .

### افكار للمستخدم الخبير

● اذا لم تظهر المخرجات الجرافيكية الخاصة بالـ object بالشكل المتوقع ، تأكد من انك قمت بتعيين قيمة لون لخاصية الـ Pen.Color التابعة للـ Canvas . اذا استمرت صورتك غير ظاهرة ، تأكد من انك ترسم فى الـ Canvas الصحيح . انتبه الى عبارات الـ with ، والتى قد تجعل العبارات تشير الى خاصية الـ Canvas فى الـ object مثل الـ bitmap بدلاً من الـ form أو الـ PaintBox المرادة .

● بسبب التغييرات فى الحاسب المكتبى للـ Windows 95 ، فقط اصبح الرسم على ايقونة - لتحريك تطبيق مصغر ، مثلاً ، غير محبذ . ولكن ، يمكنك ان تفعل هذا مع تطبيقات الـ Windows 3.1 بتعيين الـ TIcon object لخاصية الـ Application.Icon . تنحدر الـ TIcon class من الـ TGraphic - استخدم الـ TIcon بنفس الطريقة التى تستخدم بها الـ TBitmap .

● لحذف صورة مثل bitmap كبيرة مرتبطة بـ Image object، قم بتمرير nil  
للـ Assign method بخاصية الـ Picture. على سبيل المثال، تسمح العبارة  
التالية أى بيانات للـ Image1، وهو ما قد تفعله لتحفظ الذاكرة:

Image1.Picture.Assign(nil);

● ان استدعاء الـ Update method للـ form بعد الـ Invalidate يعد امراً  
اختيارياً. ولكن، يجب عليك دائماً ان تستدعي الـ Invalidate قبل  
استدعاء الـ Update. ان استدعاء الـ Update وحده لا يفعل شيئاً.  
واستدعاء الـ Invalidate (والـ Update اختياريًا) يولد ايضاً OnPaint  
event لأية PaintBox objects فى الـ form.

● لا تقم باستدعاء الـ Update أكثر من مرة، أو انك سوف تجعل برنامجك  
كسولاً. ان الـ Windows يوحد الاستدعاءات المتعددة للـ Invalidate  
ويبعث رسالة wm\_Paint عندما يكون كل شئ هادئ على جبهة الرسالة.  
استخدم الـ Update لكى يحدث تحديث فورى للشاشة.

● ان تحديد مواصفات الـ Height والـ Width للـ TBitmap تنسخ الـ bitmap  
الى الذاكرة. يمكنك ان تغير ابعاد الـ TBitmap فى أى وقت، ولكن اذا  
كانت الصورة اصغر حجماً، فإن أى pixels خارج الابعاد الجديدة يعتبر أنه  
قد فقد الى الابد.

### المشروعات التى يمكنك تجربتها

(١٣-١): حاول إدخال الـ Image، والـ Shape، والـ PaintBox فى الـ  
Panel لإنشاء toolbar جرافيكية. يمكنك استخدام هذه الـ  
objects مع الـ SpeedButtons واللوحات الفرعية العادية  
لتجهز Status lines و toolbar خاصة بتطبيق.

(١٣-٢): اكتب تطبيقاً يعرض ملف الـ Calendar.bmp الخاص بـ Delphi  
الموجود على دليل الـ C:\...\Images\Backgrnd أضف  
ايقونات سحب واسقاط يمكن للمستخدمين تحريكها فى محيط  
يوم- مثلاً، للإشارة الى عيد ميلاد أو موعد ما.



(١٣-٣): اكتب Utility يعرض كل ملفات الايقونات فى دليل ال  
C:\Delphi\Images\Icons .

(١٣-٤): اكتب برنامجاً يعرض ملف bitmap . (ملحوظة . استخدم  
(Off Screen bitmap) .

(١٣-٥): متقدم : قم بتطوير procedures الصور المتحركة التى تعرض  
bitmaps مخزنة فى Objects للـ TStringList . (ملحوظة :  
انظر الباب الثامن لمعرفة وصف owner-draw controls  
ومشروع الـ GlyphList على القرص المدمج) .

(١٣-٦): اكتب utility تستخدم الـ Animate لعرض صور Windows  
متحركة معيارية .

(١٣-٧): باستخدام التطبيق MediaPlayer كمرشد لك ، اكتب برنامج  
AudioPlayer يمكنك ان يدير ملفات الـ (WAV) .

## ملخص:

- توفر عدد من components خاصية الـ Canvas التى يمكنك استخدامها للرسم . تعتبر الـ Canvas object مشتقة من الـ TCanvas class ، والتى تضم الـ Windows Graphics Device Interface (GDI) وعناصر الـ device context لـ handle .
- لتشكيل مخرجات جرافيكية ، عين قيم لخصائص الـ Pen والـ Brush التابعة للـ Canvas . لرسم شكل معين ، استدع methods فى الـ Canvas مثل الـ Rectangle والـ Ellipse .
- استخدام الـ PaintBox لتوفير Canvas وإضافة امكانيات جرافيكية للـ object مثل الـ Panel . يمكنك ايضاً استخدام واحداً أو أكثر من الـ PaintBox objects لقصر المخرجات الجرافيكية على المنطقة المستطيلة المحددة فى الـ form .
- تعتبر الـ TGraphic هى الـ class السابقة للـ TBitmap ، والـ TIcon ، والـ TMetafile . تحتوى الـ TPicture class على الـ TGraphic object . يمكن لبرنامجك ان ينشئ TPicture objects لتخزين صور

## الباب الثالث عشر : تطوير تطبيقات الجرافيكية

جرافيكية في الذاكرة . أو ، يمكنك إضافة Image على form ، واستخدام خاصية ال Picture التابعة له لنفس هذا الغرض . هذا أيضاً ينطبق على ال controls المنحدرة من ال TGraphic مثل ال TJPEGImage .

● أن ال TPicture class والتي تتمتع بال file-handling methods مثل ال LoadFromFile وال SaveToFile ، تحدد نوع صورة الجرافيك من امتداد اسم الملف لذلك ، ويستطيع ال TGraphic object ان يقرأ ويكتب meta file ، أو bitmap ، أو ايقونة . ويمكنه أيضاً التعامل مع ال controls المنحدرة من ال class مثل ال TJPEGImage وغيرها .

● استدع ال Draw methods أو ال StretchDraw للCanvas لرسم أى TGraphic object (meta file ، أو bitmap أو ايقونة) . استخدم ال Draw لرسم الجرافيك فى احجامة المحددة . استخدم ال StretchDraw لإعادة تحديد حجم الصور لتلائم داخل المستطيل المحدد .

● توفر ال TBitmap class خاصية ال Canvas التى يمكنك استخدامها لرسم offscreen Bitmap . استدع ال CopyRect أو ال BrushCopy methods التابعة للـ PaintBox offScreen أو ال form لرسم ال offScreen Bitmap .

● قم بإنشاء Shape objects للسحب والاسقاط من خلال ال OnMouseDown ، وال OnMouseMove ، وال OnMouseUp ، كما يوضح تطبيق ال DragMe الخاص بهذا الباب .

● يستطيع ال Animate عرض AVI clips صامتة ، إما محملة من ملف قرص ، أو مختارة من احدى ال Windows clips المعيارية .

● يعمل ال MediaPlayer كـ controller لاجهزة الوسائط المتعددة المتنوعة . يوضح هذا الباب كيفية استخدام ال MediaPlayer لإنشاء متصفح ملف صور AVI .

فى واجهة التطبيق الجرافيكية ، للمستخدم تعتبر الطباعة وعرض الصور الجرافيكية موضوعات مرتبطة ببعضها البعض الى حد كبير . يشرح الباب القادم امكانيات طباعة النص والصور الجرافيكية الخاصة بـ Delphi .

\*\*\*\*\*



# الباب الرابع عشر

## تطوير تطبيقات الطباعة

محتويات هذا الباب:

• Components

• طباعة Plain-text

• الـ TPrinter class

• طباعة الجرافيك

لطباعة النص والجرافيك، تطلب منك كثير من نظم التطوير أن تستخدم أوامر الـ "escape" القديمة لـ Windows والتي يمكنها أن تجعل حتى modules الطباعة البسيطة مثل ألم الأسنان. لحسن الخط، إن الـ functions والـ procedures والـ Delphi components المتعلقة بالطباعة قد أزلت هذه الصعوبة وجعلت تطوير تطبيق الطباعة، مع مرور الوقت، شيئاً غير مؤلم.

في هذا الباب، سوف أشرح كيفية عرض dialog الإعداد للطباعة وكيفية استخدام الـ Printers unit الخاصة بـ Delphi لطباعة النص والجرافيك. سوف أقترح أيضاً تقنية لإنشاء أمر مراجعة الطباعة النهائية - وهي ميزة يجب أن توفرها كل تطبيقات الطابعات بحيث يتمكن المستخدمون من فحص المخرجات على الشاشة، بدلاً من إهدار الصفحات الاختيارية.

### :Components

فيما يلي قائمة بالـ components المتعلقة بالطباعة في Delphi:

• **PrintDialog**: استخدم هذا ال components ليستخدمه المستخدمين عند بداية عملية الطبع . يتضمن ال Windows dialog components معيارى يمكن المستخدمين من تحديد صفحات الخاصة للطبع ويوفر وصولاً لخيارات الاعداد للطابعة. ال Palette : Dialogs .

• **PrinterSetupDialog**: استخدم هذا ال components لعرض واحداً أو أكثر من dialog الاعداد المتوفرة بواسطة برامج تشغيل الطابعة المركبة . يجب ان تضم دائماً هذا ال dialog فى تطبيقات الطابعة لتعطى المستخدمين method لتشكيل الطابعات أو الاختيار من بين اجهزة الطابعة المتعددة وابواب الإخراج . ال Palette : Dialogs .

• **TPrinter**: هذا ال components غير موجود على ال VCL palette ، ولكنه معرف كclass فى ال Printers unit . من بين اعضاءها الآخرين، توفر ال TPrinter class خاصية Canvas والتي تعطيك سطح رسم WYSIWYG لطباعة النص والجرافيك ال Palette : لا يوجد .

## :Plain text

تستطيع تطبيقات Delphi ان تطبع النص بإحدى طريقتين . فى هذا الفصل ، سوف اشرح اسهل الطرق باستخدام ال Write procedures وال Writeln (تنطق write-line) التابعة للـ Pascal مع ملف إخراج ال Text . ان هذا لا يعد ملف قرص ، ولكن ملف بالمعنى العام للبرمجة للتمييز عن فيض البيانات . عندما يكون لديك نصاً فقط للطباعة ، فهذه هى التقنيات التى يجب ان تستخدمها .

**ملحوظة:** ان تقنيات ال Write وال Writeln التالية تطبع النص باستخدام ال TrueType Font الجرافيك الاخرى ، والذي يمكنك ان تضعه بنمط سميك أو مائل اذا اردت . حتى هذه التقنيات مخصصة لطباعة النص بشكل صارم ، إن هذه الطرق لا تستخدم مجموعة الرموز الافتراضية للطباعة الا اذا قمت بتركيب برنامج تشغيل طابعة نص فقط . ان بعض الطابعات الحديثة تخطط ال TrueType Font مباشرة لـ Font الطابعة الداخلية ، ولكن لكل الاغراض العملية ، يعتبر هذا الفعل شفافاً .

## الـ Printers unit

لكل module تحتاج الى إمكانيات طباعة، أضف الـ Printers unit لتعريف الـ uses الخاص بالـ module . على سبيل المثال، ابدأ مشروعاً جديداً وقم بتعديل تعريف الـ uses الخاص بالـ unit الرئيسية مثل هذا (الإضافة الجديدة هي المكتوبة بخط سميك):

uses

Windows, Messages, SysUtils, Classes, Graphics,  
Controls,  
Forms, Dialogs, Printers;

الـ Printers unit توفر الـ Printer object والتابع للـ TPrinter class . ان كل اوامر اخراج الطباعة تمر على هذا الـ object، والذي يوفر ايضاً معلومات مفيدة مثل عرض وطول الصفحة بالـ pixels.

**Tip** فكرة: قد يكون هذا واضحاً، ولكن يجب الإشارة الى أن الـ Printers unit هي جمع؛ بينما الـ Printer object مفرد. اذا واجهت مشكلة في بدء الطباعة، تأكد من أنك لم تخلط بين الكلمتين.

سوف أضف كل عضو في الـ TPrinter class مؤخراً في هذا الفصل . ولكن أولاً، لنلقي نظرة على عملية طبع بسيطة توضح كيفية استخدام الـ Printers unit والـ Printer object التابع لها . إوصل طابعتك، وقم بتشغيلها، وإتبع هذه الخطوات :

١- ابدأ تطبيقاً جديداً . إنتقل لنافذة الـ code editor، وفي القمة عند تعريف الـ uses، أضف الـ Printers الى قائمة الـ units الأخرى التي تستخدمها هذه الـ module .

٢- أضف الـ Button object من لوحة الـ Standard على الـ form واضغط الـ Button مرتين لإنشاء الـ OnClick .

٣- انسخ البرمجة الموجودة في القائمة (١٤-١) الى الـ Button1Click الـ procedure . توضح الـ code تقنيات طباعة نص أولية .

٤- قم بتشغيل البرنامج بضغط F9 . عندما تظهر نافذة البرنامج، اضغط الزر لطباعة الـ string التالي في أعلى الصفحة (إعتماداً على نوع الطباعة التي لديك، قد تكون المخرجات صغيرة جداً):

Hello printer!

القائمة (١٠-١٤) الـ `OnClick` الخاص بهذا الـ `Button` يظهر تقنيات طباعة نص أولية

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FPrn: System.Text;
begin
  AssignPrn(FPrn);
  Rewrite(FPrn);
  try
    Writeln(FPrn, 'Hello printer!');
  finally
    CloseFile(FPrn);
  end;
end;
```

لتوفير ملف يبعث إليه البرنامج مخرجات النص `procedure`، يعرف الـ `FPrn` (يمكنك أن تسميه اسماً آخر) من نوع الـ `System.Text`. قم بأداء خطوتين لإلحاق متغير الملف إلى `procedures` الإخراج الخاصة بالـ `Pascal` وقم بفتحه للعمل:

```
AssignPrn(FPrn); { Attach file for output }
Rewrite(FPrn); { Open the file }
```

الـ `Printers unit` توفر الـ `AssignPrn` لتعيين ملف المخرجات ولتوفير `Buffer` للمخرجات بالذاكرة. الـ `Rewrite` ويعتبر `Pascal procedure` معيارى يفتح ملف نص. بعد هاتين الخطوتين، يمكنك استدعاء الـ `Writeln` لطباعة `strings`. (إن أولئك الذين يعرفون الـ `Pascal` ولكنهم تطوروا سريعاً إلى الـ `Windows` قد لا يعرفوا الـ `write` والـ `Writeln`، لذا سأسترجعهم سريعاً هنا). إن استدعاء الـ `Writeln` يطبع `string` ويبدأ خطأً جديداً لتولى أية `exception` محتملة، استخدم الـ `FPrn` داخل الـ `try`:

```
try
  Writeln(FPrn, 'Plain text is plain and simple!');
  { ... insert other output statements here }
```

## الباب الرابع عشر : تطوير تطبيقات الطابعة

أغلق الطابعة ، التى تنهى مهمة الطبع وأخرج أى صفحة تامة بشكل جزئى ،  
باستدعاء ال CloseFile لتكون فى أمان ، أدخل هذه العبارة فى ال finally ، والذى  
يعتبر مضمون فى أن ينفذ حتى اذا فشل الطبع :

finally

CloseFile(FPrn);

end;

لاحظ أن ال end مطلوبة لإنهاء عبارة ال try-finally ، رغم أنه لا يوجد  
begin . يمكنك أيضاً استخدام ال WriteLn لطباعة قيم متعددة ، اعداد صحيحة ،  
متغيرات نقطة عائمة ، objects أنواع البيانات البسيطة الأخرى :

WriteLn(FPrn, 'Number of components = ', ComponentCount);

استخدم Write لطباعة النص بدون أن تبدأ سطر جديد . على سبيل المثال ،  
السطور التالية مساوية للعبارة الفورية السابقة :

Write(FPrn, 'Number of components = ');

Write(FPrn, ComponentCount);

WriteLn(FPrn); { Start new line }

ان محاولة تمرير ملف مخرجات طابعة والذى يكون قد تم تعيينه بـ  
AssignPrn للـ Read أو ال ReadLn يولد Run-time error exception .  
لا يمكنك قراءة البيانات من على جهاز مخرجات الطابعة .

### ال Fonts وال Control codes

ان ال Write وال WriteLn يستجيبان لاربعة control codes ، وهى  
مذكورة فى جدول (١٤-١) ، والتى يمكنك استخدامها لارسال اوامر للطابعة .

جدول (١٤-١) : control codes الاربعة الخاصة بالـ Write وال WriteLn

الأمور	control codes
Tab	#9
New Line	#10
Carriage Return	#13
New Page	^L



يمكنك ضم قيم الـ ASCII هذه strings أو طباعتها بعبارات Write مثل

هذه :

```
Write(FPrn, #9); { Tab }
Write(FPrn, #13); { Flush output buffer }
Write(FPrn, #10); { Flush and start new line }
Write(FPrn, ^L); { Flush and start new page }
```

تقوم الـ Write والـ WriteIn بتحديد الـ tabs الى ثمانية أضعاف متوسط عرض الرمز بالـ pixels للـ Font الحالي. عند الطباعة بـ font نسبي، هذا يعنى أنك لا تستطيع استخدام الـ tabs لمحاذاة الأعمدة الغير رقمية بالـ Write والـ WriteIn. فى معظم الـ fonts النسبية، رغم ذلك، تكون الأرقام أحادية المسافة من حيث الحجم، لذا قد تريد استخدام code لضبط الـ tab لمحاذاة الأعمدة الرقمية الخالصة. يوفر الـ Printer خاصية الـ Canvas التى، من بين مهارتها الأخرى، تحدد font الطباعة، بـ ١٠ نقاط حسب النظام الافتراضى يمكنك تغيير font الطباعة بتعيين أسمه لخاصية الـ Font.Name التابعة للـ Canvas. على سبيل المثال، إستخدم هذه العبارات لإختيار الـ TrueType Courier New font بـ ١٢ نقطة لعبارات الـ Write والـ WriteIn المتتالية :

```
with Printer.Canvas do
begin
  Font.Name := 'Courier New';
  Font.Size := 12;
end;
```

يمكنك إضافة هذا الـ code للمثال السابق فى Button1Click. إدخال السطور الخمسة المذكورة هنا بعد عبارة الـ Rewrite مباشرة (فوق الكلمة الأساسية try). والآن عندما تضغط F9 لتشغيل البرنامج وتضغط الزر، يتم طباعة النص بـ ١٢ نقطة أحادية المسافة.

**تحذير:** عند طباعة نص عادى بإستخدام التقنيات الموضحة فى هذا الفصل، إستدع دائماً الـ AssignPrn والـ Rewrite قبل تغيير قيم خاصية الـ Font فى الـ Canvas التابع للـ Printer.



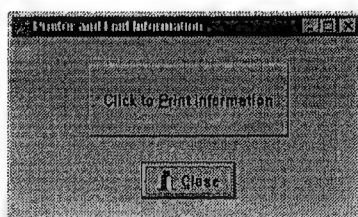
## الباب الرابع عشر : تطوير تطبيقات الطباعة

يؤثر الـ font المختار على عرض وطول character pixels، ولذلك، بتغيير عدد السطور في كل صفحة يمكنك إستخدام أى font مركب، ولكن الـ TrueType تعتبر الأفضل دائماً للطباعة لأن الـ Windows GDI يمكن أن يقدم character لأي driver، حتى لو لم يكن يستخدم الـ TrueType مباشرة.

يجب أن تعرف أيضاً كيف تستخدم تقنيات الـ Write والـ WriteIn الموضحة هنا للطباعة على طابعات النص فقط مثل تلك الأنواع القديمة، إذا كان يجب عليك التعامل مع طابعات النص فقط، إستخدم Control Panel لترتيب برنامج التشغيل الـ Windows Generic/Text Only وتشغيله. ولكن لا يوجد ضمان أن هذا سوف يعمل، وللحصول على أفضل النتائج، يجب أن تحدد للمستخدم النهائي لبرنامجك أن طابعة الجرافيك مطلوبة للطباعة مع الـ Windows.

### إحصائيات الطباعة

على القرص المدمج، إن تطبيق الـ PrnInfo الموضح فى شكل ١٤-١ يطبع إحصائيات عن إمكانيات طابعتك والـ font الحالى. توجد ملفات البرنامج على القرص المدمج فى دليل الـ Source\PrnInfo. يوضح البرنامج أيضاً كيفية تحديد عدد السطور لكل صفحة، والذي قد يكون مفيداً لبعض التطبيقات.



شكل (١٤-١): إضغط الزر الكبير للـ PrnInfo للحصول على تقرير مطبوع عن برنامج تشغيل طابعتك، وعدد السطور فى كل صفحة، ومتوسط عدد الرموز لكل سطر.

لإستخدام البرنامج، حمل ملف مشروع الـ PrnInfo فى الـ Delphi، إفتح طابعتك، إضغط F9 لتشغيل البرنامج، وإضغط الزر الكبير. وما قام الـ PrnInfo بطباعته هو عن مواصفات نظامى.

Device = HP DeskJet 720C Series on LPT1:

Font = Courier New

Font Size = 12 points

PageHeight = 3150 pixels

PageWidth = 2400 pixels

Extent.Cx = 6720 pixels

Extent.Cy = 53 pixels

Lines per page = 56

Chars per line = 80

توضيح القائمة (١٤-٢) source code لـ PrnInfo.

القائمة (١٤-٢): PrnInfo\Main.pas

unit Main;

interface

uses

SysUtils, Windows, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, Printers, StdCtrls, Buttons;

type

TMainForm = class(TForm)

PrintButton: TButton;

CloseBitBtn: TBitBtn;

procedure PrintButtonClick(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

MainForm: TMainForm;

implementation

{ \$R \*.DFM }

procedure TMainForm.PrintButtonClick(Sender: TObject);

var

FPn: System.Text;

Extent: TSize;

Metrics: TTextMetric;

I, LinesPerPage, CharsPerLine, AverageWidth: Integer;

S : String;

begin

AssignPrn(FPn);

Rewrite(FPn);

with Printer.Canvas do

begin

Font.Name := 'Courier New';

Font.Size := 12;

end;

{ Fill test string with ASCII values 32 to 255 }

try

(\* S[0] := Chr(224); // This is no longer allowed \*)

SetLength(S, 224); // Use this method instead

for I := 32 to 255 do // Fill string with test chars

S[I - 31] := Chr(I);

with Printer, Canvas do

begin

{ Determine number of lines per page }

GetTextExtentPoint(Handle, @S[1], Length(S),  
Extent);

LinesPerPage := PageHeight div (Extent.Cy + 2);

```

if PageHeight mod Extent.Cy <> 0 then
    Dec(LinesPerPage);
    { Determine average number of characters per line }
    GetTextMetrics(Handle, Metrics);
    AverageWidth := Metrics.tmAveCharWidth;
    CharsPerLine := PageWidth div AverageWidth;
    { Print the report }
    Writeln(FPrn, 'Device = ', Printers[PrinterIndex]);
    Writeln(FPrn, 'Font = ', Font.Name);
    Writeln(FPrn, 'Font Size = ', Font.Size, ' points');
    Writeln(FPrn, 'PageHeight = ', PageHeight, ' pixels');
    Writeln(FPrn, 'PageWidth = ', PageWidth, ' pixels');
    Writeln(FPrn, 'Extent.Cx = ', Extent.Cx, ' pixels');
    Writeln(FPrn, 'Extent.Cy = ', Extent.Cy, ' pixels');
    Writeln(FPrn, 'Lines per page = ', LinesPerPage);
    Writeln(FPrn, 'Chars per line = ', CharsPerLine);

end;
finally
    CloseFile(FPrn);
end;
end;

end.

```

فى المعلومات المذكورة، يساوى الـ Extent.Cx عدد الـ pixels الأفقية التى ستكون مطلوبة لطباعة الـ string الإختبارى للبرامج بإستخدام الـ font الحالى (حدد الـ Courier New بـ ١٢ نقطة). إن الـ PmInfo لا يطبع هذا الـ string - فهو يستخدم فقط لإعداد التقرير. ويساوى الـ Extent.Cy الارتفاع بالـ pixels للـ string الإختبارى. ويكون عدد السطور لكل صفحة دقيق فقط إذا قمت بطباعة كل السطور على الصفحة بنفس الـ font. وعدد الرموز فى كل سطر يكون تقريبى للـ pixels النسبية.

وتكون هذه القيمة عادة حاسمة فى تحديد كم المعلومات التى تتناسب داخل صفحة من المخرجات. تأكد من إختيار اسم font وحجمه، كما هو موضح فى القائمة، قبل الحصول على المعلومات.

لعرض إحصائيات عن font مختلف، أضف عبارات مثل التالية قبل استدعاء الـ GetTextExtentPoin مباشرة:

```
Font.Name := 'Arial';
```

```
Font.Size := 24;
```

يوضح الـ PrnInfo كيفية استدعاء الـ Windows API functions التي تتطلب الـ handle للـ printer device قم بتمرير خاصية الـ Handle الخاصة بالـ Canvas للـ Printer إلى هذا الـ parameter. على سبيل المثال، هذا يؤدي إلى استدعاء الـ GetTextExtentPoint:

```
GetTextExtentPoint(Handle, @S[1], Length(S), Extent);
```

لإتمام الـ function PChar parameter، تمرر العبارة عنوان الرمز الأول (@S[1]) للـ string الإختباري. وتستدعي عبارة مشابهة API function أخرى، وهي الـ GetTextMetrics:

```
GetTextMetrics(Handle, Metrics);
```

إستخدم هذه التقنيات وقيم الـ PrnInfo الواردة، فقط مع method طباعة النص الخاصة بالـ Write و الـ WriteIn الموضحة هنا. كما سأشرح فيما بعد في هذا الباب، فإن تقنيات الطباعة الأخرى تعطيك حكماً أفضل على النتائج، خاصة عند طباعة الجرافيك.

### طباعة قوائم الـ string

لطباعة سطور نص خاصة بطباعة الـ TStringList أو الـ TStringList، إستخدم الـ WriteIn كما هو موضح في هذا الفصل. أضف الـ Printers لأمر الـ uses بالـ module وإدخل هذا الـ code في Button OnClick لطباعة قائمة string بإستخدام menu:

```
var  
  FPrn: TextFile;  
  I: Integer;  
begin  
  AssignPrn(FPrn);
```

```
Rewrite(FPrn);
try
  for I := 0 to Memo1.Lines.Count - 1 do
    Writeln(FPrn, Memo1.Lines[I]);
  finally
    CloseFile(FPrn);
  end;
end;
```

يمكنك استخدام هذا الـ code لطباعة أى خاصية TStrings - على سبيل المثال كما هو موضح هنا أو خاصية الـ Items فى الـ ListBox.

### dialogs الطابعة

إن methods واجهة التطبيق للمستخدم للطباعة تختلف فيما بين التطبيقات، ولكن أغلب البرامج تستخدم على الأرجح واحداً من أوامر قائمة الملف التقليدية. وأوامر القائمة المعيارية الأربعة هى:

❖ **FilePage Setup** - اختياري. يعرض Dialog تطبيق والذى ينظم المسافات الرأسية، وأرقام الصفحات، والهوامش، والتحديدات الأخرى المتنوعة التى تحتاجها. لا يوجد أية Methods معيارية لتنفيذ هذا الأمر، غير أنه يجب أن يسمح للمستخدمين بإضافة مواصفات وأية بنود تكون فريدة فى مميزات الطابعة الخاصة بتطبيقك. على سبيل المثال، يمكنك أن تحت المستخدمين على إدخال string ليتم طباعته فى أعلى كل صفحة.

❖ **FilePrint Preview** - اختياري يعرض صفحات مطبوعة مقلدة فى النافذة. لا يوجد أية طرق معيارية لتنفيذ هذا الأمر، ولكن مؤخراً فى هذا الباب سوف أقترح تقنية سوف تعمل لجميع التطبيقات.

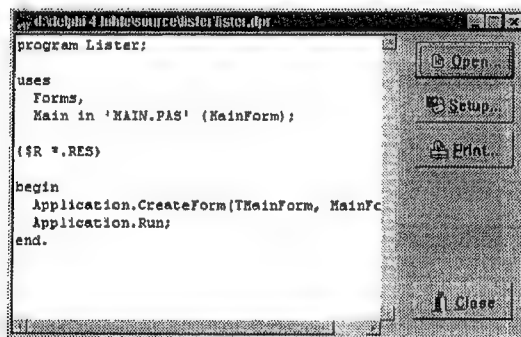
❖ **FilePrint Setup** - يعرض dialog الإعداد لبرنامج تشغيل الطابعة الحالى، والذى يتضمن زر Options يفتح dialog إضافية. هذا الأمر يجب ألا يطبع أى شئ. استخدم الـ PrinterSetupDialog لتبسيط تنفيذ هذا الأمر.

❖ **FilePrint** - يعرض الـ dialog مواصفات الطابعة للمستخدمين لكى يدخلوا عدد النسخ التى يريدونها ومتوسط عدد الصفحات. إذا ضغط المستخدم زر الـ OK

## الباب الرابع عشر : تطوير تطبيقات الطباعة

لد dialog ، يجب أن يبدأ برنامج الطباعة على الفور . وال dialog box الخاص بهذا الأمر يشمل زر Setup الذى يعرض نفس dialog الإعداد مثل أمر ال FilePrint Setup . إستخدم ال PrinterDialog لتبسيط تنفيذ هذا الأمر .

**على القرص المدمج :** يعتبر أول أمرين إختياريين . ويعتبر ال FilePrint Setup ، وال FilePrint معياريين ، بالرغم من أنه بإستطاعتك أن تستمر مع ال FilePrint... فقط لأن هذا الأمر يتيح الوصول إلى dialog إعداد برنامج التشغيل ولكن معظم التطبيقات يجب أن تنفذ على الأقل أمرى ال FilePrint Setup وال FilePrint... يوضح تطبيق ال Lister على القرص المدمج - والموضح فى شكل (١٤-٢) والقائمة (١٤-٣) - كيفية إستخدام ال PrintDialog وال PrinterSetupDialog التابعة لـ Delphi فى كتابة أوامر ال FilePrint... وال FilePrint Setup... الخاصة بالبرنامج . ويستطيع برنامج ال Lister أن يطبع أى ملف نص pas. أو txt. ، أو أى ملف نص آخر . لعرض النص والبيانات الرقمية الخاصة بالطباعة ، فإن ال Lister يسبق السطور بأرقام للسطر . ويستخدم النص المطبوع ال Courier New font ذى العشر نقاط . توجد كل ملفات ال Lister على القرص المدمج فى دليل ال Source\Lister.



شكل (١٤-٩٢) يطبع تطبيق ال Lister أى ملف نص ويعرض كيفية تنفيذ أمرى ال FilePrint Setup وال FilePrint.

القائمة ١٤-٣ : Lister\Main.pas

unit Main;

interface



uses

SysUtils, Windows, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, StdCtrls, Buttons, Printers;

type

```
TMainForm = class(TForm)
    Memo1: TMemo;
    SetupBitBtn: TBitBtn;
    PrintBitBtn: TBitBtn;
    CloseBitBtn: TBitBtn;
    OpenBitBtn: TBitBtn;
    OpenDialog1: TOpenDialog;
    PrintDialog1: TPrintDialog;
    PrinterSetupDialog1: TPrinterSetupDialog;
    procedure OpenBitBtnClick(Sender: TObject);
    procedure SetupBitBtnClick(Sender: TObject);
    procedure PrintBitBtnClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;
```

var

MainForm: TMainForm;

implementation

{ \$R \*.DFM }

```
procedure TMainForm.OpenBitBtnClick(Sender: TObject);
begin
    with OpenDialog1 do
        if Execute then
            begin
```

```
        Memo1.Lines.LoadFromFile(FileName);
        Caption := Lowercase(FileName);
    end;
end;

procedure TMainForm.SetupBitBtnClick(Sender: TObject);
begin
    PrinterSetupDialog1.Execute;
end;

procedure TMainForm.PrintBitBtnClick(Sender: TObject);
var
    FPrn: System.Text; { Printer output text file }
    I: Integer;         { Memo1.Lines index }
    LCol: Integer;      { Line number column width }
begin
    if PrintDialog1.Execute then
    begin
        AssignPrn(FPrn); { Direct Write/Writeln to FPrn }
        Rewrite(FPrn);   { Open printer output file }
        try
            Printer.Canvas.Font := Memo1.Font; { Use Memo's font }
            with Memo1, Lines do
            begin { Set line number column width for file size: }
                if Count < 10 then LCol := 1    { 0 .. 9 lines  }
                else if Count < 100 then LCol := 2 { 10 .. 99 lines }
                else if Count < 1000 then LCol := 3 { 100 .. 999 lines }
                else LCol := 4; { You must be kidding }
                for I := 0 to Count - 1 do
                begin
                    Write(FPrn, I + 1:LCol, ', '); { Print line number }
                    Writeln(FPrn, Lines[I]);      { Print line }
                end;
            end;
        finally
            CloseFile(FPrn); { Close printer output file }
```

```
end;
end;
end;
```

end.

لتنفيذ أمر الـ FilePrint Setup...، كما فى برنامج الـ Lister، أضف الـ PrinterSetupDialog على الـ form. يتم تمثيل الـ object بأيقونة، والتي لا تظهر فى وقت التشغيل. وداخل الـ OnClick للـ Button أو لأمر قائمة، أدخل هذه العبارة:

```
PrinterSetupDialog1.Execute;
```

أو، يمكنك القيام بفعل آخر إذا أغلق المستخدمون الـ dialog الإعداد بإختيار زر الـ OK له، بالرغم أن هذا ليس ضرورياً:

```
if PrinterSetupDialog1.Execute then
{ ... take action if user selected OK }
```

قم بتنفيذ أمر الـ FilePrint بأسلوب مشابه، ولكن أضف الـ PrintDialog (وهذا أيضاً ليس له ظهور فى وقت التشغيل).

توضح القائمة (١٤-٤) التخطيط العام الـ OnClick الخاص بالأمر. وداخل الـ try، قم بإضافة عبارات Write أو الـ WriteLn كما يوضح الـ procedure الـ PrintBitBtnClick لتطبيق الـ Lister.

القائمة (١٤-٤): نفذ أمر الـ FilePrint باستخدام هذا النوع من الـ code.

```
if printDialog1. Execute then
begin
AssignPrn(FPrn);
Rewrite(FPrn);
try
{ ... Write text to FPrn }
finally
CloseFile(FPrn);
end;
end;
```

## الباب الرابع عشر : تطوير تطبيقات الطباعة

في أغلب الحالات ، إنك تستخدم نفس ال font للطباعة الذي تستخدمه لعرض النص على الشاشة ، عند استخدام component مثل ال Memo لعرض نص ، عين ال Font الخاص بال object لل Canvas الخاصة بال Printer بعبارة مثل التالية ، ولكن تأكد من فتح ملف المخرجات قبل هذا :

```
Printer.Canvas.Font := Memo1.Font;
```

وإستخدم أية معلومات متوفرة في ال PrintDialog أمر يرجع إليك . بعد إستدعاء ال Execute ، إستخدم قيم خصائص ال Copies ، ال FromPage ، ال ToPage ، وقيم خصائص أخرى في ال print loop الخاصة بك :

```
with printDialog1 do
for Copy := 1 to Copies do
for Page := FromPage to ToPage do
{ ... Print pages here }
```

## ال TPrinter class :

قبل الإنتقال للفصل التالي ، والذي يشرح كيفية طباعة الجرافيك ونص ال WYSIWYG ، إقرأ التحليلات التالية لخصائص ال TPrinter methods إنك تحتاج لفهم جيد لهذه الأشياء عندما تحتاج عملية الطباعة الخاصة بك أن تمتد إلى مابعد ال Write procedures وال WriteLn الخاصة ب Delphi استخدام ال Printer object للإشارة إلى كل ال methods والخصائص المذكورة في هذا الفصل . على سبيل المثال ، التعبير Printer.Canvas يؤدي إلى الوصول إلى ال Canvas الخاص بال Printer لإستخدام هذه الخصائص وال method ، أضف ال Printers إلى أمر ال uses لل module الخاصة بتطبيقك .

## خصائص ال TPrinter

فيما يلي قائمة بخصائص ال TPrinter :-

● **Aborted** - هذه الخاصية تصبح True إذا ما قام المستخدم بإنهاء (عمل) الطباعة . مثلاً بإغلاق ال Print Manager . إفحص متغير ال Boolean هذا في ال print loop الخاصة بك لتحديد ما إذا كان ال TPrinter Abort method قد تم إستدعاءه ، وفي هذه الحالة ، يجب أن تنتهي ال loop فوراً دون إستدعاء ال EndDoc .

• **Canvas** - لطباعة الجرافيك ونص ال WYSIWYG، عين قيم خصائص ال Pen وال Brush والخصائص الأخرى لل Canvas وإستدع method الخاص بها، تماماً مثلما تفعل لعرض الجرافيك.

• **Capabilities** - هذه الخاصية تذكر مجموعة إمكانيات برنامج تشغيل جهاز الطباعة. إنها مجموعة من قيم ال pcCopies، وال pcOrientation، وال pcCollation.

• **Copies** - تساوى عدد النسخ التى تم طبعتها.

• **Fonts** - تعرض قائمة TStrings بكل ال fonts التى توفرها الطباعة الحالية. يمكنك دائماً الطباعة بإستخدام ال TrueType fonts. ولكن هذه لن تكون مشكلة لل Windows 95, 98, NT.

• **Handle** - قم بتمرير هذه القيمة لأى Windows API function تتطلب ال device context (HDC) handle على سبيل المثال، ال GetText Metrics.

• **Orientation** - لهذه الخاصية إحدى قيمتين: ال poPortrait أو poLandscape. يمكنك تعيين هذه القيم قبل الطباعة، أو يمكنك فحصها لتحديد parameters الصفحة.

• **PageHeight** - يساوى طول صفحة طباعة واحدة بال pixels. وهذه القيمة يمكن أن تتغير بشكل كبير فيما بين الطابعات.

• **PageNumber** - تساوى رقم الصفحة الحالية. ومعنى الصفحة يرجع إلى تطبيقك. فإن ال Printer يزيد ال PageNumber فى كل مرة تستدعى فيها ال NewPage method. لطباعة النص العادى، يتم زيادة ال PageNumber عندما يبدأ ال WriteIn صفحة جديدة.

• **PageWidth** - يساوى عرض صفحة طباعة واحدة بال pixels. هذه القيمة قد تختلف كثيراً بين الطابعات.

• **PrinterIndex** - يساوى Index اسم الطباعة الحالية فى قائمة ال string Printers.

• **Printing** - يساوى True أثناء عملية الطبع.

## الباب الرابع عشر : تطوير تطبيقات الطباعة

● **Title** - يحتوى string، والذي يمكنك تعيينه، ليحدد مهمة الطبع فى ال Print Manager أو للشبكة.

### الـ TPrinter methods

فيما يلى قائمة بالـ TPrinter methods :-

● **Abort** - يمكنك إستدعاء الـ Abort لإنهاء عملية الطباعة العاملة. على سبيل المثال، يمكنك عرض modeless dialog box قبل أن يقوم تطبيقك بإضافة الـ print loop الخاصة بك. إستخدم flag عام لتشير إذا ما كان كذلك، إستدع الـ Printer. ABORT وإخرج من الـ printer-output loop (لا تستدع أيضاً الـ EndDoc) إن إستدعاء الـ Abort يحدد الـ Printer.Aborted flag بـ True.

● **BeginDoc** - إستدع هذا قبل بدء مهمة طباعة جديدة. لا يجب عليك إستدعاء الـ BeginDoc عند إستخدام تقنيات النص العادى للـ Write والـ WriteLn الموضحة فى بداية هذا الباب.

● **EndDoc** - إستدع هذا بعد إنهاء عملية الطباعة. إن الـ EndDoc يفرغ الـ Buffer المخرجات ويخرج الصفحة الأخيرة إذا لزم الأمر. لا تستدع الـ EndDoc عند إستخدام تقنيات الـ plain-text للـ Write والـ WriteLn. كذلك، لا تستدع الـ EndDoc بعد إنهاء عملية الطبع بإستدعاء الـ Printer.Abort (لكن هذا ليس خطيراً فى النسخ 3 و4 لـ Delphi كما كان فى Delphi 1 & 2).

● **NewPage** - إستدع هذا الـ procedure فى أى وقت لبدء صفحة جديدة. إن معنى ومحتوى الصفحة يرجع إليك وعلى برنامجك أن يوفره. إن الـ NewPage يقوم بتجميع الـ Buffer المخرجات ودفع الصفحة الحالية.

**تحذير:** لا تقم بإنشاء object من الـ TPrinter class. إن الـ unit Printer توفر الـ Printer object من نوع بيانات هذه الـ class ليكون معداً للإستخدام.

Caution

## طباعة الجرافيك

إن تقنيات طباعة النص الموضحة فى الفصل السابق تمثل منطقة وسط بين سهولة الإستخدام والنتائج. عندما تحتاج إلى مزيداً من التحكم فى parameters

المخرجات - لتحصل على نص موضوع بدقة على الصفحة، مثلاً، أو للتعامل مع الـ tabs للحصول على fonts نسبية - يمكنك استخدام الـ method التالية. في هذا الفصل، سوف أوضح كيفية طباعة الـ forms، الجرافيك، نص الـ WYSIWYG، والـ bitmaps والتطبيق على الـ font، الخاص بهذا الفصل، وهو الـ FontSnap، يقدم أيضاً تقنيات ذات هدف عام لمعاينة شاشة المخرجات المطبوعة.

### طباعة الـ forms:

قد تريد أن تتضمن أمراً يطبع لقطات من مظهر الـ form على الشاشة. هذا سهل - فقط استدع الـ form print method. لا يجب عليك أن تستدع أى من الـ TPrinter method، أو إضافة Printers لتعريف الـ uses الخاص بالـ module. لطباعة الـ form، أضف الـ PrintDialog على الـ form (هذا لا يظهر فى وقت التشغيل أو على المخرجات المطبوعة)، ثم قم بتنفيذ الـ OnClick للـ Button أو لأمر القائمة بعباراة مثل هذه:

```
if PrintDialog1.Execute then
    Print;
```

طباعة الـ form ما بالأسم، استخدم هذه العبارة:

```
AboutBox.Print;
```

والـ AboutBox هو خاصية الـ Name التى تعينها للـ form. داخلياً، يقوم الـ form Print method بنسخ الـ client area الخاصة بالنافذة الحالية إلى الـ bitmap بعيدة عن الشاشة. ثم يقوم الـ print method بعد ذلك بطباعة هذه الـ bitmap بإستدعاء الـ Windows API StretchDIBits function. قم بتعين واحدة من ثلاث قيم لخاصية الـ form PrintScale لتحديد كيف يتم قياس المخرجات. يمكنك أن تفعل هذا فى وقت التشغيل أو فى الـ Object Inspector.

● **poNone-No**. إن حجم الـ form المطبوعة يعتمد على الـ resolution الطباعة. إن إستخدام هذا الخيار لطباعة الـ forms على طابعات الليزر ينتج عنه نوافذ صغيرة تكون حادة ولكن غير مقروءة. إنك لن تستخدم هذا الخيار غالباً.

● **poPrintToFit** - يحدد حجم الصفحة. تملأ الـ form المطبوعة الصفحة فى إتجاه واحد على الأقل (أفقياً غالباً). هذا الخيار يعتبر أكثر الخيارات إهداراً للحبر

## الباب الرابع عشر : تطوير تطبيقات الطابعة

الطباعة ، ولكن عندما تزيد لل form أن تملأ الصفحة ، وهذا هو الخيار الصحيح للاستخدام .

● **poProportional** - يحدد باستخدام خاصية ال pixels في كل بوصة (أو ال pixels-per-inch) للطباعة . هذا يؤدي إلى مخرجات ذات أفضل شكل ، والتي تبدو ، من مسافة الرؤية العادية ، شبيهة في الحجم لمظهر ال form على الشاشة . هذا لا يعني أن الأحجام تتماشى بشكل دقيق ؛ إن المخرجات المطبوعة تعتبر منطقة وسط مقبولة بين القراءة والمسافة .

### طباعة object الجرافيك

إن تقنيات طباعة الجرافيك ونص ال WYSIWYG . وهو النص المطبوع ك object الجرافيك - تتطلب تقنيات مختلفة عن طباعة النص العادي بعبارة ال Write وال Writeln . إنك تستخدم نفس ال Printers unit وال Printer object ، ولكنك تستدعي TPrinter class method لبدء وإنهاء عملية الطباعة . يجب عليك تعريف معنى الصفحة لتطبيق ورسم كل output object هذا يتطلب مزيداً من العمل ، ولكنه يوفر أعلى سيطرة على النتائج .

على القرص المدمج : توضح القائمة ١٤-٥ الخطوات الأساسية المطلوبة لطباعة محتويات ال Image . (يوجد هذا النص في ملف Print1.pas على القرص المدمج في دليل ال (Source\PrintMisc) . لا يجب عليك استخدام ال component ، ولكنه أفضل لهذا البرنامج . والخطوات الثلاث الأساسية هي :-



١- استدع Printer BeginDoc procedure لبدء الطباعة .

٢- استدع Canvas method الخاص بال Printer .

على سبيل المثال ، استدع Draw لطباعة TGraphic ، bitmap أو أيقونة أو (meta file) ، أو يمكنك استدعاء methods مثل ال Ellipse وال Rectangle . للحصول على تحكم أفضل للخطأ ، قم بهذه الخطوات داخل ال try .

٣- استدع ال EndDoc عندما تنتهي من الطباعة . للحصول على تحكم أفضل للخطأ ، استدع ال EndDoc في ال finally .



القائمة (٥-١٤)، أتبع هذه الخطوة العامة لطباعة الجرافيك.

```
procedure TForm1. printClick (Sender: TObject);
begin
  Printer.BeginDoc;
  try
    Printer.Canvas.Draw(0, 0, Image1.Picture.Graphic);
  finally
    Printer.EndDoc;
  end;
end;
```

لتجربة ال procedure الموجود في القائمة ١٤-٥، أضف Image و Button على ال form. إن حجم ال Image غير مهم.

إستخدم خاصية ال Image Picture لتحميل أى ملف. إضغط bitmap مرتين لإنشاء OnClick له، وإستبدل ال procedure الناتج بالبرمجة الموجودة في القائمة قم بتشغيل البرنامج بضغط F9 ثم إضغط الزر لطباعة ال bitmap لا تهتم كثيراً إذا كان حجم المخرجات صغيراً هذا يحدث لأنه، إذا لم تكن قد قست الصور المطبوعة طبقاً ل resolution الطابعة، فإن النتائج تستخدم ال resolution الافتراضى للطابعة. هذا يمثل مشكلة كبيرة خاصة مع طابعات الليزر، التى توفر resolution كبيراً جداً عن شاشات العرض الحالية. فعند ٣٠٠ نقطة لكل بوصة، قد تكون مخرجات الجرافيك صغيرة جداً بحيث لا تقرأ. لقياس الصور بحيث تتماشى بشكل أفضل مع ال resolution المخرجات، إستدع ال Windows API GetDeviceCaps وأطلب قيم ال logPixelsX وال logPixelsY ل device printer context handle. هذا ال pixel لكل بوصة للجهاز، والتى، عند تقسيمها على خاصية ال form PixelsPerInch، تقاس الصورة بحجم مناسب.

إن الرقم الحقيقى لل pixels لكل بوصة فى جهاز معين قد تختلف عن القيمة المنطقية لعدة أسباب، واحداً منها أن ال Windows يعرض نص الشاشة أكبر منه فى واقع الأمر، لذا تكون أحجام النقاط الصغيرة مقروءة على شاشات منخفضة ال resolution.

على القرص المدمج: تقوم القائمة ١٤-٥ بقياس الصورة المطبوعة طبقاً للطابعة. إستبدل OnClick للبرنامج الإختبارى بهذه القائمة، أعد التشغيل والطبع. يجب أن يكون حجم الصورة الآن أقرب لمظهرها على



## الباب الرابع عشر : تطوير تطبيقات الطباعة

الشاشة . مرة أخرى ، هذا لا يعنى أن الشاشة والأحجام المطبوعة تتطابق بالضبط ؛ فقط إنها تبدو متشابهة من مسافات الرؤية العادية . هذا النص للقائمة ١٤-٦ موجود على القرص المدمج فى دليل SourcePrintMisc ، فى ملف Print2.pas .

القائمة (١٤-٦) : استخدم هذا الـ code لقياس صورة طبقاً لـ resolution للطابعة.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ScaleX, ScaleY: Integer;
  R: TRect;
begin
  Printer.BeginDoc;
  with Printer do
  try
    ScaleX :=
      GetDeviceCaps(Handle, logPixelsX) div PixelsPerInch;
    ScaleY :=
      GetDeviceCaps(Handle, logPixelsY) div PixelsPerInch;
    R := Rect(0, 0, Image1.Picture.Width *
      ScaleX,
      Image1.Picture.Height * ScaleY);
    Canvas.StretchDraw(R,
      Image1.Picture.Graphic);
  finally
    EndDoc;
  end;
end;
```

إن الـ procedure الموجود فى القائمة ١٤-٦ يبدأ بإثنين من متغيرات العدد الصحيح ، وهى ScaleX و ScaleY ، لقيم تساوى الـ pixels لكل بوصة خاصة بالطابعة التى تم الحصول عليها بواسطة الـ GetDeviceCaps مقسمة على خاصية الـ PixelsPerInch form . هناك خاصية واحدة لأن نظام عرض الـ VGA مربعة ، وعدد الـ pixels لكل بوصة يجب أن يكون واحداً رأسياً وأفقياً . ولا يعتبر نفس الشئ صحيحاً بالضرورة لكل أجهزة المخرجات ، وهذا هو السبب فى أن الـ GetDeviceCaps يوفر قيم لمحاور الـ x والـ y .

بعد تحديد الـ ScaleX والـ ScaleY، يقوم الـ procedure بإنشاء الـ TRect، R، لأبعاد المخرجات. وبضرب الـ Picture.Width والـ Picture.Height التابعة للـ Image فى عوامل القياس يتم تعديل المستطيل ليتناسب بشكل مقبول. أخيراً، الـ StretchDraw ترسم الـ bitmap بالإشارة إلى خاصية الـ Graphic للـ Picture.

**ملحوظة:** تبدأ عملية الطباعة فعلياً عندما يستدعى البرنامج الـ EndDoc أو الـ NewPage. إن الرسم بخاصية الـ Printer Canvas لا ينتج أى مخرجات.



### طباعة الـ bitmap، والإيقونات، والـ metafile:

**على القرص المدمج:** لا يجب عليك إستخدام الـ Image لطباعة الجرافيك. يمكنك إستخدام التقنية الموضحة فى القائمة ٦-١٤ لطباعة الـ bitmap والأيقونات والـ meta file. قم بإنشاء الـ TPicture (أنظر الباب السابق لمعرفة التعليمات) وقم بتمرير خاصية الـ Graphic التابعة له للـ Draw methods أو الـ StretchDraw بالـ PrinterCanvas. توضح القائمة ٧-١٤ كيفية إستخدام هذه التقنية لطباعة ملف الـ Sample.bmp على القرص المدمج فى دليل الـ Source\Data. قم بتعديل string اسم المسار كما هى الحاجة فى الإستدعاء للـ Load Fram File. يوجد نص هذا الـ procedure على القرص المدمج فى دليل الـ Source\PrintMisc فى ملف الـ Print2.pas إن القائمتين ٦-١٤ و ٧-١٤ تنتجان نفس النتائج. وأى تقنية تستخدم تعتمد على ما إذا كنت تريد إستخدام الـ Image أو إنشاء الـ TPicture تحت تحكم البرنامج. إن عرض الـ Image هو الأسهل غالباً، فإذا كنت لا تستطيع أن تحدد، إستخدم الـ method الموجود فى القائمة ٦-١٤. ولكن، القائمة ٧-١٤ أكثر فاعلية فى إستخدامها للذاكرة لأن الـ bitmap object موجود فقط داخل نطاق الـ event nandler.



القائمة ٧-١٤ إستخدم هذا الـ code لطباعة

الـ bitmap أو أيقونة أو metafile دون إستخدام الـ Image.

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

```

P: TPicture;
ScaleX, ScaleY: Integer;
R: TRect;

begin
  P := TPicture.Create;
  try
    // Modify the pathname string in the following statement
    // to load any .bmp, .ico, or metafile:
    P.LoadFromFile('D:\Delphi 4
    Bible\Source\Data\Sample.bmp');
    Printer.BeginDoc;
    with Printer do
      try
        ScaleX :=
          GetDeviceCaps(Handle, logPixelsX) div
          PixelsPerInch;
        ScaleY :=
          GetDeviceCaps(Handle, logPixelsY) div
          PixelsPerInch;
        R := Rect(0, 0, P.Width * ScaleX, P.Height *
          ScaleY);
        Canvas.StretchDraw(R, P.Graphic);
      finally
        Printer.EndDoc;
      end;
    finally
      P.Free;
    end;
  end;
end;

```

### طباعة أشكال الجرافيك :

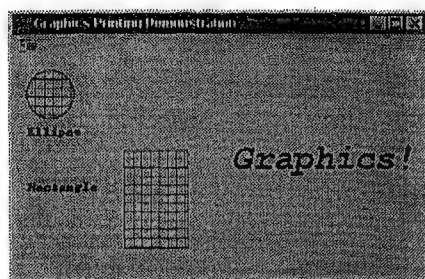
على القرص المدمج : لعرض وطباعة object الجرافيك ، أكتب  
 procedure يرسم الجرافيك والنص على Canvas يتم تمريره على أنه  
 argument . إستدع procedure للطباعة على ال Canvas الخاص بالـ



. Printer

إستدع نفس الـ procedure في OnPaint الخاص بالـ form لعرض  
 الجرافيك . والحيلة هي أن تقيس الصورة المطبوعة فتخرج على حجم معقول للورقة

وعلى الشاشة. على القرص المدمج، يوضح تطبيق الـ PrintGr فى دليل الـ Source\PrintGr كيفية طباعة وعرض object الجرافيك والنص يوضح شكل (١٤-٣) عرض الـ PrintGr. توضح القائمة (١٤-٨) source code البرنامج. قم بتشغيل البرنامج، إختبر أمر الـ File\Print للطباعة.



شكل (١٤-٣): يوضح الـ PrintGr إحدى طرق عرض وطباعة objects الجرافيك والنص.

القائمة (١٤-٨): PrintGr\Main.pas

unit Main;

interface

uses

SysUtils, Windows, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, Menus, Printers;

type

TMainForm = class(TForm)

MainMenu1: TMainMenu;

File1: TMenuItem;

Print1: TMenuItem;

N1: TMenuItem;

Exit1: TMenuItem;

PrintDialog1: TPrintDialog;

procedure Print1Click(Sender: TObject);

procedure FormPaint(Sender: TObject);

```
procedure Exit1Click(Sender: TObject);
private
procedure PaintGraphics(C: TCanvas;
    ScaleX, ScaleY: Integer);
public
{ Public declarations }
end;

var
    MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.PaintGraphics(C: TCanvas;
    ScaleX, ScaleY: Integer);
var
    R: TRect;
    P: TPoint;

function ScalePoint(X, Y: Integer): TPoint;
begin
    Result := Point(X * ScaleX, Y * ScaleY);
end;

function ScaleRect(L, T, R, B: Integer): TRect;
begin
    Result :=
        Rect(L * ScaleX, T * ScaleY, R * ScaleX, B *
            ScaleY);
end;

begin
    with C do
        begin
```

```

Pen.Color := clBlue;
Brush.Color := clRed;
Brush.Style := bsCross;
Font.Name := 'Courier New';
Font.Size := 8;
Font.Style := [fsBold, fsItalic];
R := ScaleRect(12, 12, 57, 57);
Ellipse(R.Left, R.Top, R.Right, R.Bottom);
R := ScaleRect(100, 85, 160, 174);
Rectangle(R.Left, R.Top, R.Right, R.Bottom);
P := ScalePoint(12, 60);
TextOut(P.X, P.Y, 'Ellipse');
P := ScalePoint(12, 110);
TextOut(P.X, P.Y, 'Rectangle');
Font.Size := 24;
Font.Style := [fsBold, fsItalic];
P := ScalePoint(200, 75);
TextOut(P.X, P.Y, 'Graphics!');
end;
end;
procedure TMainForm.Print1Click(Sender: TObject);
var
  ScaleX, ScaleY: Integer;
begin
  if PrintDialog1.Execute then
    Printer.BeginDoc;
  try
    ScaleX := GetDeviceCaps(Printer.Canvas.Handle,
      logPixelsX) div PixelsPerInch;
    ScaleY := GetDeviceCaps
      (Printer.Canvas.Handle,
        logPixelsY) div PixelsPerInch;
    PaintGraphics(Printer.Canvas, ScaleX,
      ScaleY);
  finally
    Printer.EndDoc;
  end;
end;

```

```

end;
end;

procedure TMainForm.FormPaint(Sender: TObject);
begin
  PaintGraphics(Canvas, 1, 1);
end;

procedure TMainForm.Exit1Click(Sender: TObject);
begin
  Close;
end;

end.

```

يستدعى البرنامج الـ `PaintGraphics procedure` لعرض `object` ونصه . يعرف الـ `procedure` ثلاثة `parameters` : `TCanvas` ، وأثنين من الـ `Integers` وهما ؛ الـ `ScaleX` والـ `ScaleY` ، لقياس الإحداثيات .

وتقوم الـ `functions` بأداء القياس . تقوم الـ `ScalePoint` بضرب الـ `x` والـ `y` فى عوامل القياس وترجع سجل الـ `TPoint` بنفس الشيء ، ولكن ترجع سجل الـ `TRect` . ويوضح الـ `PaintGraphics` كيفية إستخدام هذه الـ `functions` . على سبيل المثال ، لتلوين زر ببيضاوى ينفذ البرنامج هذه العبارات :

```

R := ScaleRect(12, 12, 57, 57);
Ellipse(R.Left, R.Top, R.Right, R.Bottom);

```

إن إستدعاء الـ `ScaleRect` يضبط الإحداثيات لتشمل الـ `resolution` الأكبر لمعظم الطابعات . ولكن هذه ليست الطريقة الوحيدة لأداء القياس - يوضح الفصل التالى تقنية أخرى .

### إنشاء أمر `preview` للطبع

إذا لم تتطابق `resolutions` الطابعة والشاشة بالضبط ، فمن المستحيل إنتاج الـ `object` الجرافيكية الموجودة على الشاشة ، ولكن بواسطة برمجة حذرة و `font` `TrueType` قابلة للقياس ، تكون النتائج جيدة لكثير من التطبيقات .

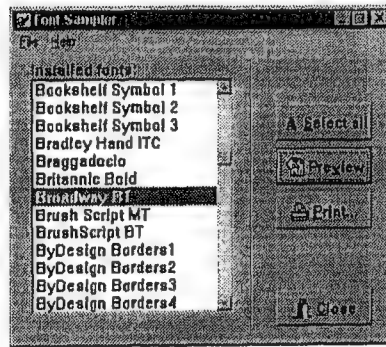


على القرص المدمج: إن البرنامج الأخير فى هذا الباب يعرض مخرجات الـ WYSIWYG، ويوضح مزيداً من تقنيات الطبع، مثل multipage printing وكيفية تنفيذ نافذة preview الطباعة.



والبرنامج الذى يسمى FontSamp، يعرض قائمة من الـ fonts لتجربة البرنامج، قم بتحميل ملف مشروع الـ FontSamp.dpr من على القرص المدمج فى دليل الـ Source\FontSamp. إضغط F9 للتشغيل وإختر واحداً أو أكثر من أسماء الـ fonts ويمكنك أيضاً ضغط الـ Select all button لتختار كل الـ font ثم تضغط Print لطبع الصفحات لكل font مختار بأحجام وأساليب متنوعة. إضغط Preview لترى الصفحات قبل الطباعة. يوضح شكل ١٤-٤ العرض الرئيسى للبرنامج. إختر واحداً أو أكثر من أسماء الـ fonts من مربع القائمة.

إضغط Print لطباعة عينات كل font. إضغط Preview لرؤية الصفحات قبل الطبع. يوضح شكل ١٤-٥ نافذة الـ Preview.

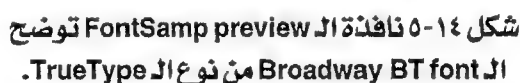


شكل رقم (١٤-٤): العرض الرئيسى لـ FontSamp.

**ملحوظة:** تعرض نافذة الـ preview مراجعة الشكل النهائى لد FontSamp TrueType fonts مثل الـ Arial والذى يتميز بدقة أكثر من الـ bitmap fonts مثل الـ Courier. زد حجم نافذة الـ preview للحصول على أفضل النتائج.



تقوم الـ TBitmap class الخاصة بـ Delphi بتبسيط الخطوات اللازمة لإنشاء نافذة preview. يقوم الـ FontSamp بإنشاء الـ TBitmap. وعرض وطول محدد بنفس النسبة مثل الـ ١١ × ٨١ / ٢ بوصة للصفحة.



وغالباً من الناحية العملية، من الصعب تطابق المخرجات المطبوعة بدقة في bitmap صغيرة. وتعتبر الـ Fonts غير الـ TrueType مزعجة ولا يتم إعادة تحديد حجمها بشكل جيد. وإحدى الخطط التي تبدو ناجحة، والتي لا تتطلب أشهر طويلة، هي برمجة كل الإحداثيات كقيم floating-point لتمثل البوصات. هذه الخطة تحافظ على برنامج منظم وبسيط. على سبيل المثال، لطباعة نص في منتصف صفحة، بفرض أن البعد الرأسى يساوى ١١ بوصة، فإنك تختار إحداثية الـ y مساوية لـ ٥, ٦ بوصة. كل ما تحتاجه هو بعض الـ functions البسيطة لتحويل البوصات إلى pixels مؤسوسة على الـ resolution المنطقي للـ Canvas.

وهناك ثلاثة تراكيب فى FontSamp تنفذ هذه الخطوة وهى : Main ، Preview ، و DrawPage . توضح القائمة ٩-١٤ OnClick التابعين للـ Main module .

```

القائمة ٩-١٤: نرى الـ Print والـ Preview للـ FontSamp\Main.pas
procedure TMainForm.PreviewBitBtnClick(Sender: TObject);
begin
    if PreviewForm.ShowModal = mrOk then
        PrintBitBtn.Click; { Preview's print button }
    end;

    { This is the procedure that prints the pages }
    procedure TMainForm.PrintBitBtnClick(Sender: TObject);
    var
        PpiX, PpiY, Page, FirstPage, LastPage: Integer;

        { Initialize PrintDialog1 object }
        procedure InitPrintDialog;
        begin
            with PrintDialog1 do
                begin
                    MinPage := 1;
                    MaxPage := FontListBox.SelCount;
                    FromPage := MinPage;
                    ToPage := MaxPage;
                end;
            end;

        { Initialize printing variables }
        procedure InitParameters;
        begin
            { Do our own scaling based on Page width and height.
            This

```

```
seems to be more reliable than GetDeviceCaps. }  
  PpiX := Trunc(Printer.PageWidth / 8.5);  
  PpiY := Trunc(Printer.PageHeight / 11.0);  
  { Set FirstPage and LastPage }  
  if PrintDialog1.PrintRange = prAllPages then  
  begin  
    FirstPage := 1;  
    LastPage := FontListBox.SelCount;  
  end else  
  begin  
    FirstPage := PrintDialog1.FromPage;  
    LastPage := PrintDialog1.ToPage;  
  end;  
end;
```

```
begin  
  InitPrintDialog;  
  if PrintDialog1.Execute then  
  begin  
    Printer.BeginDoc;  
    try  
      InitParameters;  
      for Page := FirstPage to LastPage do  
      begin  
        DrawOnePage(Printer.Canvas, FontListBox,  
          Page,  
            Printer.PageWidth, Printer.PageHeight,  
            False,  
              PpiX, PpiY);  
        if page < LastPage then  
          Printer.NewPage;  
        end;  
      finally  
        Printer.EndDoc;
```

```
end;
end;
```

```
end;
```

ال Preview OnClick يعرض ال ShowModal . إذا رجعت هذه ال function بـ mrOK ، يكون المستخدم قد ضغط زر ال Preview Print من أجل الطباعة . يستدعى Print Click method, procedure .

يقوم ال Print OnClick بإستدعاء procedures منطقيين لبدء dialog الطباعة و parameters أخرى . حدد خصائص ال MinPage وال MaxPage لل PrintDialog1 بمدى الصفحة . فى هذه الحالة ، هناك font واحد لكل صفحة ، لذا يكون أكبر عدد من الصفحات مساوياً لل FontListBox.SelCount حدد ال FromPage وال ToPage بنفس هذه القيم حتى تظهر فى نوافذ ال dialog الطبع . لبدء الطباعة ، يحدد ال InitParameters procedure متغيرات ال PpiX وال PpiY التى تكون مساوية لعدد ال pixels المنطقية لكل بوصة لحجم الصفحة المعنية ذات ال ٨, ٥ × ١١ بوصة .

إن عمل الحسابات بهذه الطريقة يبدو أكثر دقة من إستدعاء ال GetDeviceCaps . ويقوم ال procedure بفحص خاصية ال PrintRange لل PrintDialog1 . فإذا كانت مساوية للثابت prAllPages ، يكون المستخدم قد ضغط ال All radio button الخاص بال dialog ؛ وإلا ، يكون المستخدم قد قام بتعديل مدى الصفحة ويمكنك الحصول على هذه القيم من ال ToPage وال FromPage كما تم التوضيح .

وبعد كل هذا ، يعتبر البرنامج مستعداً للطباعة . وتتولى for loop هذا الأمر بإستدعاء procedure ، وهو ال DrawOnePage (فى DrawPage module) مع arguments متنوعة . وأكثر أهمية هى الأولى ، Printer.Canvas ، والتى توجه المخرجات إلى الطابعة . أما ال Printer.PageWidth وال Printer.PageHeight فيعطيان ال DrawOnePage حدود المخرجات لصفحة واحدة . إن إستدعاء ال Printer.NewPage ينقلك إلى صفحة جديدة بعد الرسم .

## الباب الرابع عشر : تطوير تطبيقات الطباعة

إن ال Preview module فى القائمة (١٤-١٠) تستدعى ال DrawOnePage لتحاكى الصفحات المطبوعة . وهذا ال module قصيرة نسبياً ، لذا ذكرتها كاملة هنا .

القائمة ١٤-١٠ : FontSampl\Preview.pas

```
unit Preview;
```

```
interface
```

```
uses
```

```
SysUtils, Windows, Messages, Classes, Graphics,  
Controls, Forms, Dialogs, StdCtrls, Buttons,  
ExtCtrls,  
DrawPage;
```

```
type
```

```
TPreviewForm = class(TForm)
```

```
  ToolBar: TPanel;
```

```
  LeftPageSB: TSpeedButton;
```

```
  RightPageSB: TSpeedButton;
```

```
  PrintSB: TSpeedButton;
```

```
  CloseSB: TSpeedButton;
```

```
  procedure FormCreate(Sender: TObject);
```

```
  procedure FormResize(Sender: TObject);
```

```
  procedure FormPaint(Sender: TObject);
```

```
  procedure FormDestroy(Sender: TObject);
```

```
  procedure CloseSBClick(Sender: TObject);
```

```
  procedure PrintSBClick(Sender: TObject);
```

```
  procedure FormActivate(Sender: TObject);
```

```
  procedure LeftPageSBClick(Sender: TObject);
```

```
  procedure RightPageSBClick(Sender: TObject);
```

```
private
```

```
PreBits: TBitmap;    { Preview bitmap with  
canvas }
```

```
PpiX, PpiY: Integer; { Logical pixels per inch  
}
```

```

Page: Integer;          { One font sampler per page }
    procedure InitGlobals; { Initialize global
variables }
    public
        FontListBox: TListBox; { Reference to form's
ListBox }
end;

var
    PreviewForm: TPreviewForm;

implementation

uses Main;

{$R *.DFM}

const
    border = 10;    { Top and bottom preview bitmap borders }

{ Create form and a bitmap to represent the preview page }
procedure TPreviewForm.FormCreate(Sender: TObject);
begin
    FontListBox := nil;
    PreBits := TBitmap.Create;
end;

{ Initialize global variables and window size }
procedure TPreviewForm.InitGlobals;
begin
    PreBits.Width :=
        ClientWidth div 2; { Bitmap width = @bf1/2 client
width }
    PreBits.Height :=
        Round(1.3 * PreBits.Width); { 1.3 = 8@bf1/2 x 11
ratio }
    PpiX :=

```

```

Round(PreBits.Width / 8.5); { Logical pixels per inch }
PpiY :=
Round(PreBits.Height / 11.0); { Logical pixels per
inch }
if WindowState <> wsMaximized then { Adjust window
bottom }
ClientHeight :=
ToolBar.Height + PreBits.Height + border * 2;
end;

```

{ Tip: OnResize is called before OnActivate, but only if the form is NOT maximized, in which case FormResize is never called. Don't use OnResize as your only display initializer-also initialize in OnActivate. }

```

procedure TPreviewForm.FormResize(Sender: TObject);
begin
  InitGlobals;
  DrawOnePage(PreBits.Canvas, FontListBox, Page, {Redraw
page}
  PreBits.Width, Height, True, PpiX, PpiY);
  Invalidate;
end;

```

{ Because the program does its own scaling, we can call Draw instead of StretchDraw as some previewers do. This keeps the display fast and keeps the text looking as WYSIWYG as possible }

```

procedure TPreviewForm.FormPaint(Sender: TObject);
begin
  Canvas.Draw(ClientWidth div 4,
  ToolBar.Height + border, PreBits);
end;

```

```

procedure TPreviewForm.FormDestroy(Sender: TObject);
begin
  PreBits.Free;

```



end;

```
procedure TPreviewForm.CloseSBClick(Sender: TObject);
begin
    ModalResult := mrCancel;
end;
```

```
procedure TPreviewForm.PrintSBClick(Sender: TObject);
begin
    ModalResult := mrOk;
end;
```

```
{ This procedure prepares the FontListBox, and it draws
the first page (or a blank if no font is selected). The
procedure also enables and disables the toolbar SpeedButtons }
procedure TPreviewForm.FormActivate(Sender: TObject);
begin
```

```
{ If you don't assign a ListBox to PreviewForm.FontListBox,
this statement picks up the ListBox from the parent form. }
```

```
    if FontListBox = nil then
```

```
        FontListBox := MainForm.FontListBox;
```

```
{ Draw first page }
```

```
    Page := 1;
```

```
    InitGlobals;
```

```
    DrawOnePage(PreBits.Canvas, FontListBox, Page,
        PreBits.Width, Height, True, PpiX, PpiY);
```

```
{ Enable / disable speed buttons in toolbar }
```

```
with FontListBox do
```

```
begin
```

```
    LeftPageSB.Enabled := SelCount > 1;
```

```
    RightPageSB.Enabled := SelCount > 1;
```

```
    PrintSB.Enabled := SelCount > 0;
```

```
end;
```

```
end;
```

```
{ Display previous page }
```

=====

```

procedure TPreviewForm.LeftPageSBClick(Sender: TObject);
begin
    if Page > 1 then
    begin
        Dec(Page);
        DrawOnePage(PreBits.Canvas, FontListBox,
            Page,
            PreBits.Width, Height, True, PpiX, PpiY);
        Invalidate;
    end;
end;

{ Display next page }
procedure TPreviewForm.RightPageSBClick(Sender: TObject);
begin
    if Page < FontListBox.SelCount then
    begin
        Inc(Page);
        DrawOnePage(PreBits.Canvas, FontListBox,
            Page,
            PreBits.Width, Height, True, PpiX, PpiY);
        Invalidate;
    end;
end;

end.

```

تعرف ال Preview module متغيرات متعددة خاصة بال TPreviewForm class . ويمثل ال PreBits ، وهو أحد ال TBitmap object ، صفحة واحدة من ال preview . وتعتبر ال PpiX وال PpiY هي ال pixels المنطقية لكل بوصة في هذه ال bitmap . أما ال Page فهي رقم الصفحة وال FontListBox هو مرجع لل form الأم FontListBox . ويقوم ال Procedure InitGlobals ببدء هذه المتغيرات .

وتقوم ال Preview form بإنشاء ال PreBits bitmap في ال OnCreate ويوضح ال InitGlobals procedure كيفية أداء بعض الحسابات الضرورية

لصفحة المخرجات المقلدة إن تعيين قيم للـ `PreBits.Width` والـ `PreBits.Height` يحدد حجم `bitmap` بالنسبة لنافذة الـ `Preview`، ولكن بالنسبة لـ  $11 \times 81 / 2$  بوصة كحجم الصفحة. يحدد البرنامج الـ `PpiX` والـ `PpiY` بالـ `pixels` لكل بوصة لنفس حجم هذه الصفحة - وهذا لا يساوى الـ `pixels` لكل بوصة على الشاشة أو على الـ `form` ولكن يساوى الـ `bitmap` التى تحاكى صفحة المخرجات. أخيراً، إذا لم يتم تكبير النافذة، يعين البرنامج قيمة معدلة لخاصية الـ `ClientHeight` للـ `form`. هذا ينقل زر النافذة حتى تصبح صفحة الـ `Preview bitmap` مرئية تماماً دائماً.

حاول أن تعيد تحديد حجم نافذة الـ `Preview` وأنظر كيف يحاول زر النافذة أن يبحث عن مستواه.

إننتقل إلى الـ `FormPaint OnPaint`. لعرض صفحة الـ `Preview` يمرر البرنامج `PreBits` إلى الـ `Canvas Draw method`. هذا يجعل المخرجات سريعة. ويستطيع أمر `Preview` للطبع أن يشكل `bitmap` بالحجم الحقيقى ثم يستخدم الـ `StretchDraw` بعد ذلك لعرض هذه الـ `bitmap` فى نافذة. وهذا أسهل فى البرمجة، ولكنه يعطى نتائج ضعيفة.

ولأن البرنامج يستخدم الـ `Draw`، فيجب عليه أن يعيد رسم كل صفحة عندما:

\* تظهر نافذة الـ `Preview` لأول مرة.

\* يتغير حجم النافذة.

\* يطلب المستخدم صفحة مختلفة.

إن الـ `OnResize`، `FormResize`، يحدد متغيرات عمومية طبقاً لحجم النافذة الحالى، ويستدعى الـ `DrawOnePage` لتشكيل مخرجات الـ `preview`، للرسم على الـ `bitmap` البعيدة عن الشاشة، يمرر البرنامج الـ `PreBits.Canvas` للـ `DrawOnePage` ثم يستدعى بعد ذلك الـ `Invalidate`، مما يجعل الـ `Windows` يصدر رسالة الـ `wm_Paint` للنافذة هذا يؤدي للـ `OnPaint`، والذى، كما رأيت، يرسم الـ `bitmap` بإستدعاء الـ `Canvas.Draw method` (تلك الموجودة فى الـ `Canvas` التابع للـ `form` وليس التابع للـ `bitmap`).

عندما يتم تنشيط ال Preview form لأول مرة، ينشئ ال OnActivate بعض القيم مثل ال FontListBox reference ثم تستدعي بعد ذلك ال DrawOnePage لتكون صفحة المخرجات الأولى. ليس من الضروري أن تستدعي ال Invalidate event لإيجاد ال OnPaint، لأن هذا يحدث بصورة تلقائية بعد ال OnActivate. إن ال OnActivate يقوم أيضاً بإبطال وتشغيل ال SpeedButtons الخاصة بنافذة ال Preview اعتماداً على عدد ال fonts المختارة (ال FontListBox.SelCount).

يقوم ال FormActivate procedure بتعيين ال FontListBox لل MainForm FontListBox وهذا التعيين لا يؤدي إلى إنشاء list box آخر. استخدم nil، كما هو موضح هنا ك flag، للإشارة إذا ما كان مرجعاً مثل ال FontListBox قد تم بدئه. على سبيل المثال، يمكنك إدخال اثنين أو أكثر من ال list box مع مجموعات fonts مختلفة فيهم وعين ال reference لهم بـ PreviewForm.FontListBox.

أو يمكنك أن تجعل ال module تفحص هل ال FontListBox محدد بـ nil، في هذه الحالة تنتقي ال FontListBox الخاص بال form الأم طبقاً للنظام الافتراضي للحاسب. (يتم إنشاء ال FontListBox بـ nil في ال OnCreate الخاص بال form). وهذا المثال جيد على البرمجة الدفاعية - إن ال module تعمل بشكل صحيح بغض النظر عما إذا كانت هناك module أخرى تنشئ متغير ال FontListBox العام.

أخيراً، إن ال SpeedButtons OnClick الخاصة بالصفحة التالية والسابقة لل tools bac توجد في ال Preview module. إن ال procedures تقوم بزيادة أو تقليل متغير ال Page وتستدعي ال DrawOnePage لصنع ال bitmap.

كما أوضحنا، إن استدعاء ال Invalidate يصر على حدوث ال OnPaint event، الذي يرسم ال bitmap الجديدة على ال Canvas الخاص بال Preview form.

كما أدركت حتى الآن، أن ال DrawOnePage يربط المخرجات الجرافيكية لهذا البرنامج مع بعضها البعض. وتوضح القائمة ١٤-١١ ال module ال DrawPage التي تنفذ هذا ال procedure.

القائمة ۱۴-۱۱: FontSamp\DrawPage.pas

```
unit Drawpage;

interface

uses SysUtils, Graphics, StdCtrls;

{ Call DrawOnePage to form each sampler page either
during
printing or for previewing with an offscreen bitmap. }

procedure DrawOnePage(
  Canvas: TCanvas;      { Printer or TBitmap Canvas for preview
  }
  FontListBox: TListBox; { Fonts with multiple
  selections }
  Page,          { Page number (FontList selection
  index) }
  PageWidth,      { Unscaled page width in
  pixels }
  PageHeight: Integer; { Unscaled page height in
  pixels }
  Previewing: Boolean; { True if previewing; else
  printing }
  PpiX, PpiY: Integer  { Pixels per inch on X- and Y-
  axes }
  );

implementation

uses Main;

var
  C: TCanvas;
  FontName, HeaderName: String;
  PixelsPerInchX, PixelsPerInchY: Integer;
  Preview: Boolean;
```

```

{ Return selected font at index }
function SelectedFont(ListBox: TListBox;
  Index: Integer): String;
var
  I: Integer;
begin
  with ListBox do
    for I := 0 to Items.Count - 1 do
      if Selected[I] then
        begin
          Dec(Index);
          if Index <= 0 then
            begin
              Result := Items[I];
              Exit;
            end;
          end;
          Result := 'System';
        end;
end;

{ Assign font name, style, and size to Canvas font }
procedure SetFont(const Name: String; Style: TFontStyles;
  Size: Integer);
begin
  { Adjust point size for preview page's logical pixels per
  inch relative to the form's actual pixels per inch. This allows
  the program to draw into the bitmap with TextOut, and then
  display the bitmap in real size with Canvas.Draw. Some print
  previewers use StretchDraw, which produces relatively poor
  results. }
  if Preview then
    Size := Round(Size *
      (PixelsPerInchY / MainForm.PixelsPerInch));
  { Assign parameters to Canvas C Font property }
  C.Font.Name := Name;
  C.Font.Style := Style;

```

```

C.Font.Size := Size;
end;

{ Return pixel width of Name in inches }
function InchWidth(const Name: String): Double;
begin
    Result := C.TextWidth(Name);
    Result := Result / PixelsPerInchX;
end;

{ Return pixel height of Name in inches }
function InchHeight(const Name: String): Double;
begin
    Result := C.TextHeight(Name);
    Result := Result / PixelsPerInchY;
end;

{ Write string S at inch coordinates X and Y }
procedure TextAtInch(X, Y: Double; const S: String);
var
    Px, Py: Integer;
begin
    Px := Round(X * PixelsPerInchX);
    Py := Round(Y * PixelsPerInchY);
    C.TextOut(Px, Py, S);
end;

{ Draw a line at inch coordinates X1, Y1, X2, Y2 }
procedure LineAtInch(X1, Y1, X2, Y2: Double);
var
    Px1, Py1, Px2, Py2: Integer;
begin
    Px1 := Round(X1 * PixelsPerInchX);
    Py1 := Round(Y1 * PixelsPerInchY);
    Px2 := Round(X2 * PixelsPerInchX);
    Py2 := Round(Y2 * PixelsPerInchY);

```

```
C.MoveTo(Px1, Py1);
C.LineTo(Px2, Py2);
end;
```

```
{ Draw header at top of page }
procedure DrawHeader(const Name: String);
var
  S: String[24];
begin
  SetFont(HeaderName, [fsBold], 12);
  TextAtInch(0.5, 0.5, Name);
  SetFont(HeaderName, [fsItalic], 12);
  S := 'Font Sampler by Tom Swan';
  TextAtInch(8.0 - InchWidth(S), 0.5, S);
  LineAtInch(0.5, 0.5, 8.0, 0.5);
end;
```

```
{ Draw footer at bottom of page }
procedure DrawFooter(Page: Integer);
begin
  SetFont(HeaderName, [], 12);
  TextAtInch(0.5, 10.5, 'Page ' + IntToStr(Page));
end;
```

```
{ Draw sample character set (ASCII 32-255) }
procedure DrawCharacterSet;
var
  H: Double;
  procedure DrawOneLine(J, K: Integer);
  var
    I: Integer;
    S: String;
  begin
    S := "";
    for I := J to K do
      S := S + Chr(I);
```



```

TextAtInch(0.5, H, S);
  H := H + InchHeight('M');
end;
begin
  SetFont(HeaderName, [fsBold], 12);
  TextAtInch(0.5, 1.4, 'Character set:');
  SetFont(FontName, [], 10);
  H := 1.5 + InchHeight('M');
  DrawOneLine(32, 80);
  DrawOneLine(81, 129);
  DrawOneLine(130, 178);
  DrawOneLine(179, 227);
  DrawOneLine(228, 256);
end;

{ Draw sample text in 8, 14, and 24 point sizes }
procedure DrawPointSamples;
var
  H, M: Double;
  procedure DrawOneSample(Pts: Integer);
  begin
    SetFont(HeaderName, [fsBold], 12);
    TextAtInch(0.5, H, IntToStr(Pts) + ' points:');
    M := InchHeight('M');
    H := H + M;
    SetFont(FontName, [], Pts);
    TextAtInch(0.5, H, 'AaBbCc 1234567890');
    H := H + M * 2;
  end;
begin
  H := 4.0;
  DrawOneSample(8);
  DrawOneSample(14);
  DrawOneSample(24);
end;

```

```
{ Draw normal, italic, bold, and bold-italic samples }
procedure DrawNormBoldItal;
```

```
var
```

```
    H, M: Double;
```

```
    procedure DrawOneLine(const S: String; Style:
TFontStyles);
```

```
    begin
```

```
        SetFont(HeaderName, [fsBold], 12);
```

```
        TextAtInch(0.5, H, S);
```

```
        M := InchHeight('M');
```

```
        H := H + M;
```

```
        SetFont(FontName, Style, 12);
```

```
        TextAtInch(0.5, H, 'AaBbCc 1234567890');
```

```
        H := H + M * 2;
```

```
    end;
```

```
begin
```

```
    H := 7.0;
```

```
    DrawOneLine('Normal 12 points:', []);
```

```
    DrawOneLine('Italic 12 points:', [fsItalic]);
```

```
    DrawOneLine('Bold 12 points:', [fsBold]);
```

```
    DrawOneLine('Bold Italic 12 points:', [fsBold,
fsItalic]);
```

```
end;
```

```
{ Printing and preview code calls this procedure to draw
each page. See declaration at top of file for descriptions
of the parameters. }
```

```
procedure DrawOnePage(Canvas: TCanvas; FontListBox: TListBox;
```

```
    Page, PageWidth, PageHeight: Integer; Previewing:
Boolean;
```

```
    PpiX, PpiY: Integer);
```

```
begin
```

```
{ Save some parameters in global variables for easy access }
```

```
    C := Canvas;
```

```
    C.Pen.Color := clBlack;
```

```
    PixelsPerInchX := PpiX;
```

```

PixelsPerInchY := PpiY;
Preview := Previewing;
{ Draw the font samples on the Canvas (Printer or Preview) }
with Canvas do
begin
  FillRect(ClipRect); { Erase page }
  if (FontListBox = nil) or (FontListBox.SelCount <
  1) then
    Exit; { Display / print blank page if no font
    selected }
  FontName := SelectedFont(FontListBox,
  Page);
  HeaderName := 'Arial';
  DrawHeader(FontName); { These statements draw
  the }
  DrawFooter(Page); { header, footer, and font
  samples }
  DrawCharacterSet;
  DrawPointSamples;
  DrawNormBoldItal;
end;
end;

end.

```

إن تقرير كيفية تنفيذ procedures الرسم الخاصة ببرنامجك أمر يرجع إليك، ولكنني أخصصها دائماً لـ module منفصلة. بالإضافة إلى أن البرنامج أسهل في الحفظ، فإم فصل عبارات الجرافيك قد ساعدتني على حمل العديد من البرامج لنظم تشغيل متعددة، والتي يكون لها أوامر رسم مختلفة. إن module DrawPage لديها public procedure واحد، وهو الـ DrawOnePage، والذي كما رأيت، يقوم الـ FontSamp بإستدعائه لطباعة صفحات البرنامج في تطبيقاتك، قم بتصميم modules الرسم و preview قبل تنفيذ أوامر الطباعة الخاصة بالبرنامج. هذا يوفر الأوراق ويجبرك على أن تفكر جيداً في كيفية قياس المخرجات للـ resolutions المختلفة. ولأنني إتبعته هذه الخطوة في كتابة الـ FontSamp، فإن أوامر الطباعة الخاصة به عملت من المرة الأولى

## الباب الرابع عشر : تطوير تطبيقات الطباعة

التي جربتها فيها . (حسناً، ربما كانت من المرة الثالثة، ولكن على أية حال، إن أوامر الطباعة قد بدت أسهل في الإنشاء لأننى قد حللت المشكلات الصعبة في ال DrawPage modules وال Preview) إننى لم أشرح كل ال module code ال DrawPage، والتي لا تحتوى أى عبارات طباعة. ولكن، تكمن أهمية ال module فى أنها ترسم على ال Canvas، بغض النظر عن نوعه. لذلك، يستطيع البرنامج تمرير ال Canvas الخاص بال Printer لل DrawOnePage ليطبعه، أو ال Canvas الخاص بال PreBits bitmap لرسم صفحات offscreen preview.

إن البرمجة ستصبح منظمة إذا كانت العبارة السابقة صادقة بنسبة ١٠٠٪. ولكن، من الناحية العملية، لا يكون التوصل إلى مخرجات ال WYSIWYG بهذه السهولة. لقد إحتجت إلى flag، وهو ال previewing، لأحدد ما إذا كان ال mod-ule قد طبع أو تم عرضه إن ال SetFont Procedure يستخدم هذا ال flag لتقليل حجم نقطة ال flag عند ال preview، وهذا أمر ضرورى لأن النص سوف يتم تحديده بشكل طبيعى لقياس النافذة (وليس قياس ال bitmap) وللتغلب على هذه المشكلة يستطيع البرنامج رسم fonts فى أحجامها الطبيعية ويستدعى ال StretchDraw bitmap ولكن، كما ذكرت، تكون النتائج ضعيفة ويتم عرضها ببطء.

### افكار للمستخدم الخبير

\* إذا كان النص لا يطبع بحجمه المتوقع، تأكد من أن ال code تستدعى ال Print.Begin Doc قبل تغيير ال Printer Canvas Font والخصائص الفرعية له.

\* يجب عليك فى النهاية أن تستدعى ال CloseFile لمتغير ال System.Text الذى تم تمريره لل AssignPrn. إن محاولة تعيين ملف مخرجات ثانى دون إغلاق الملف المعين حالياً يؤدي إلى exception.

\* عند طباعة plain text، إنتظر حتى تتغير خاصية ال PageNumber التابعة لل Printer بعد كل إستدعاء لل WriteLine. عندما تتقدم ال PageNumber، تكون قد بدأت صفحة جديدة. بدلاً من حساب عدد السطور لكل صفحة كما يقترح هذا الباب، يمكنك إستخدام هذه التقنية لكتابة رأس سطر فى أعلى كل صفحة جديدة.

\* عين string لخاصية ال Printer Title لتعريف مهمة الطباعة فى ال Windows Print Manager وعلى عناوين صفحة الشبكة .

\* لإنشاء abort printing dialog ، إستخدم form منفصلة وإستدع Show method لها ليعرضها . إجعل ال form تحدد المتغير العام بـ Ture ك flag يشير إلى أن المستخدم قد ضغط ال Abort . إبحث عن هذا ال flag داخل ال printer output loop لبرنامجك وإذا كان Ture flag ، إستدع ال Printer.Abort method لإلغاء الطباعة . لا يجب عليك إستدعاء ال EndDoc فى هذه الحالة .

\* لفحص ما إذا كانت عملية الطبع قد تم إبطالها (على سبيل المثال ، بواسطة ال print loop التى تستدعى ال Printer.Abort) ، إفحص ال flag Read only Printer.Aborted .

## المشروعات التى يمكنك تجربتها

١٤-١ : قم بتحسين برنامج العرض Lister بإضافة أمر قائمة لإختيار ال font والنمط والحجم ، ولجعل أرقام السطور إختيارية . يمكنك أيضاً تنفيذ أمر ال FilePage Setup لتشكيل العناوين ، أرقام الصفحات ، وما إلى ذلك . إطبّع هذه البنود كما تفعل مع العناصر الأخرى بإستخدام ال Write وال WriteIn . (ملحوظة : إستخدم ال method الموضح بواسطة تطبيق ال PrnInfo لتحسب عدد السطور فى كل صفحة) .

١٤-٢ : إن تغيير ال fonts فى ال PrnInfo يجعل البرنامج يطبع تقريره بنفس هذا ال font ، مما قد يسبب عدة مشاكل - على سبيل المثال ، إذا إخترت حجم نقطة كبير أو بنط رمزى راجع هذه الميزة بكتابة procedure يعرض إحصائيات الطباعة لأى font ونمط وحجم نقطة . ثم ، إستخدم ال procedure لطباعة التقرير بال Font المعيارى .

## الباب الرابع عشر : تطوير تطبيقات الطباعة

٣-١٤ : أكتب utility لتحديد عدد السطور لكل صفحة لكل font مركب . إطبّع تقريراً عن هذه المعلومة .

٤-١٤ : قم بتصميم برنامج طباعة تشخيص يطبع غط إختياري ، يعتبر هذا البرنامج مقيداً في فحص حدة الحافة ، مظهر النص ، إعتدال الخط ، ومعلومات أخرى مفيدة .

٥-١٤ : متقدم . قم بتصميم وتنفيذ موثق مشروع Delphi الذي يطبع كل ملفات الـ pas . في الدليل الحالي وكذلك ملف مشروع dpr . وكذلك إطبّع أيقونات وملفات bitmap .

### ملخص :

\* إستخدم الـ PrintDialog والـ PrinterSetupDialog لحث المستخدمين على الطباعة وتوفير الوصول إلى إعدادات برنامج تشغيل جهاز الطباعة المركبة .

\* إن الـ TPrinter ، الذي لا يوجد على لوحة الـ VCL ، يوفر إمكانيات طباعة . أضف الـ Printer لأمر الـ uses الخاص ببرنامجك . يمكنك عندئذ تعيين قيم لخصائص الـ Printer العام وإستدعاء methods مثل الـ Printer.BeginDoc والـ Printer.NewPage . إنك لا تنشئ أبداً object من الـ TPrinter class المتوفر بالـ Printers unit .

\* توجد طريقتان للطباعة : لطباعة نص عادي ، إستخدم ملف الـ Text مع الـ Write procedures والـ WriteIn الخاصة بـ Pascal . لطباعة الجرافيك ونص الـ WYSIWYG ، إستخدم الـ Printer Canvas . إستدع الـ BeginDoc . لبداً عملية الطباعة . إرسم على الـ Canvas كما تفعل لعرض الجرافيك في نافذة . إستدع الـ NewPage للإنتقال إلى صفحة جديدة . إستدع الـ EndDoc لإنهاء الطباعة .

\* لطباعة form إستدع الـ Print method الخاص بها ، والذي يحدد مساحة الـ form client area على الـ offscreen bitmap . يستخدم الـ method تقنيات الـ BeginDoc والـ EndDoc التابعة للـ Printer لطباعة الناتج .

\* قم بإنشاء أمر preview للطبع وذلك بفصل عبارات مخرجات الجرافيك في الـ module ، أو بواسطة procedure مثل الـ DrawOnePage في تطبيق الـ

FontSamp . للطباعة ، قم بتمرير الـ Printer Canvas إلى procedure . لرسم صفحات preview مقلدة ، قم بتمرير Canvas خاص بالـ TBitmap ولرسم bitmap باستخدام الـ Canvas.Draw method form (وليس الـ bitmap) .

إن الـ Multiple Document Interface ، أو الـ MDI ، كانت واحدة من الميزات التابعة للـ Windows والمختلف عليها فيما بين واضعي البرامج منذ اليوم الأول لها . إن الـ MDI ليست مناسبة لكل البرامج ، ولكنها مفيدة في إنشاء نوافذ متعددة الصفحات بإستخدام واجهة تطبيق معيارية ، كما هو موضح في الباب التالي .

# الباب الخامس عشر

## تطوير تطبيقات الـ MDI

### محتويات هذا الباب:

- Components.
- أسس برمجة الـ MDI.
- Child windows.
- تقنيات MDI أخرى.

إن مطوري الـ Windows يستمتعون بعلاقة الحب/الكراهية للـ MDI، أو الـ Multiple Document Interface. في الواقع، إن أغلب واضعي البرامج يحبون أن يكرهوا الـ MDI. عندما تحتاج إطار عمل لإدارة وثائق متعددة، مع قالب للبيانات واحد لكل نافذة، فإن الـ MDI توفر واجهة تطبيق معيارية يسهل استخدامها، وبمساعدة Delphi، الأكثر مباشرة للبرنامج.

إن هذا الباب يشرح كيفية استخدام Delphi forms لإستخدام لإنشاء نوافذ الـ MDI الرئيسية والـ child. سوف أشرح أيضاً موضوعات مثل كيفية إنشاء قائمة Window بأوامر إدارة النافذة، وكيفية تقسيم الـ forms إلى classes فرعية لإنشاء child window، كيفية إضافة عناوين نافذة لقائمة Window، وكيفية دمج قوائم النافذة الرئيسية والصغيرة.

### : Components

فيما يلي بعض الـ Delphi components لتطوير تطبيقات الـ MDI:

\* TForm - هذا الـ component وهو نفسه كالذي تم إستخدامه في تطبيقات الـ Single Document Interface (SDI)، غير موجود على لوحة الـ



VCL . إن تطبيقات الـ MDI تستخدم objects من الـ TForm class للنوافذ الرئيسية والـ child windows . حدد خاصية الـ FormStyle لنافذة رئيسية بـ fsMDIForm . للـ Child windows ، حدد هذه الخاصية بـ fsMDIChild . وإتبع التعليمات الموجودة في هذا الباب لإنشاء Child window في وقت التشغيل - إستجابة لأمر الـ FileOpen ، مثلاً . Palette : لا يوجد .

\* MainMenu - إن نافذة الـ form الرئيسية لكل تطبيق MDI يجب أن يكون لها MainMenu object ، والذي تم تقديمه في الباب الخامس إن أغلب الـ MDI MainMenu لها عناصر قائمة File و Window ، بالرغم من أنك حر في تسمية قوائمك بما تشاء . ويمكن أيضاً لنوافذ الـ MDI الصغيرة MainMenu objects للدمج في قائمة النافذة المرئية الـ Standard: Palette .

**ملحوظة:** إنطق الـ MDI على أنها "I", "D", "M", حتى تتفادى الخلط بينها وبين الـ MIDI ، وهو واجهة تطبيق لآلة موسيقية . إن الـ MDI والـ MIDI لا علاقة لهما ببعضهما .

Note

## أسس برمجة الـ MDI

إن كل تطبيق MDI له ثلاثة أجزاء أساسية :-

\* form النافذة الرئيسية للـ MDI .

\* وثيقة واحدة أو أكثر للـ child window MDI form .

\* القائمة الرئيسية للـ MDI .

على عكس ما يحدث في برمجة الـ Windows التقليدية ، تأخذ Delphi form object محل الـ MDIframe المعيارية والـ Client window . في العادي ، تكون نافذة الإطار هي المرئية ؛ والـ client windows تعتبر نوع من الشريك الصامت الذي يتولى العمليات العامة ، ينشئ النوافذ الـ child ، ويؤدي خدمات الرسالة . في تطبيقات Delphi ، مازالت الـ client window والإطار موجودة ، ولكن نادراً ما تستخدمهما . ولجميع الأغراض العملية ، يمكنك أن تعامل نافذة الإطار الـ client على أنهما نافذة واحدة ، مقدمة لبرنامجك على أنها form النافذة الرئيسية . وتعتبر نوافذ الـ Document الصغيرة forms أيضاً ، ولكن على عكس

## الباب الخامس عشر : تطوير تطبيقات الـ MDI

نوافذ مثل الـ AboutBoxdialog التي تضيفها إلى الـ module، لا تستطيع نوافذ الـ MDI أن تخرج خارج حظيرتها. فهي مقيدة أن تظهر داخل الـ client area للنافذة الرئيسية. عندما تقوم بتصغير نافذة صغيرة، تعرض أيقونتها داخل النافذة الرئيسية، وليس في الـ Windows Start bar.

وباستثناء هذه الاختلافات، تعتبر نوافذ الـ MDI الصغيرة والكبيرة شبيهة بـ forms التطبيق ذات النافذة الواحدة في تطبيقات الـ MDI، يمكنك إضافة toolbars، status line، جرافيك و component objects، وتستخدم كل تقنيات البرمجة الأخرى لـ Delphi. ولكن تعتبر الـ MDI غاية في التميز والإفادة عندما تستخدم في تطبيقات تعمل بوئات متعددة.

**ملحوظة:** تعتبر الـ MDI عادة نظام تعامل مع ملف، ولكن لا يجب على نوافذ التطبيق الصغيرة أن ترتبط بملفات قرص. يمكنك أيضاً استخدام الـ MDI لإنشاء تطبيقات متعددة النوافذ من أنواع أخرى. والمصطلح «وثيقة» الذي يستخدم في هذا الباب يشير إلى أى معلومات يمكن عرضها في نافذة، وليس بالضرورة بيانات في ملف قرص وثيقة.

### النافذة الرئيسية للـ MDI:

لتابع هذه الخطوات لإنشاء النافذة الرئيسية لتطبيق الـ MDI:

١- أبدأ تطبيقاً جديداً.

٢- عين Name مثل الـ MainForm.

٣- حدد خاصية الـ FormStyle بـ fsMDIForm.

٤- لحفظ المشروع في دليل جديد. إجعل اسم الـ unit بـ Main.pas module، واسم الـ MDITest إذا أردت الإستمرار).

إن الـ form الرئيسية فقط هي التي يمكن أن تحدد خاصية الـ FormStyle بـ fsMDIForm ويمكن أن يكون هناك نافذة واحدة فقط مثل هذه لكل تطبيق. لضمان أن الـ form يتم إنشاؤه بصورة تلقائية، اختر الـ ProjectOptions، وتحقق من أن الـ MainForm موضحة في Main form list ومذكورة تحت الـ forms Auto-create هذا هو كل ما تحتاج أن تفعله لإنشاء النافذة الرئيسية لتطبيق الـ

MDI. ويمكنك أن تنتقل الآن للفصل التالي لإنشاء قائمة برنامجك ونوافذه الـ child.

**ملحوظة:** لتجربة التقنيات الموجودة في هذا الباب، أبدأ تطبيقاً جديداً باستخدام أمر القائمة FileNew Application، أو استخدم الـ FileNew وإختر الـ Application في مشروع MDI جديد مع قوائم افتراضية، و Child form، event handlers مؤخراً في هذا الباب، سوف أوضح كيفية استخدام مشروع الـ MDI، ولكن لأغلب برامج العرض الموجودة في هذا الباب يمكنك إنشاء تطبيق ذو نافذة واحدة خالية ثم أتبع الخطوات التالية.

### إضافة Child MDI forms

يحتاج كل تطبيق MDI أيضاً إلى Child form واحدة على الأقل و unit. إتبع هذه الخطوات لإضافة نافذة Child MDI إلى تطبيق MDI جديد:

١- إبدأ تطبيقاً جديداً بإختيار أمر الـ FileNew Application. قم بتغيير خاصية الـ Form1 Name، باستخدام الـ Object Inspector، إلى MainForm. و قم بتغيير خاصية الـ MainForm FormStyle إلى fsMDIForm.

٢- إختار FileSave All (أو إضغط زر السرعة Save All). قم بتغيير اسم الـ Unit1.pas إلى Main.pas وإحفظه. قم بتغيير الـ Project1.dpr إلى MDITest.dpr وإحفظه.

٣- إختار FileNew Form أو إضغط زر New form هذا يؤدي إلى إنشاء form جديد يسمى Form1 حسب النظام الافتراضى. (إنه يسمى Form1 وليس Form2 لأنك قمت بتغيير اسم الـ form الأخرى لتصبح MainForm). أعد تحديد حجم نافذة الـ Form1 لتجعلها أسهل في الإختيار (إنما في البداية تكون في نفس حجم الـ MainForm وتغطي هذه النافذة في الغالب).

٤- استخدم الـ Object Inspector لتغيير خاصية الـ Name في Form1 إلى ChildForm. حدد الـ FormStyle بـ fsMDIChild.

٥- إختار FileSave All (أو إضغط زر Save All). عليك إعطاء اسم ملف للـ ChildForm unit module هذا الاسم من Unit1.pas،

## الباب الخامس عشر : تطوير تطبيقات الـ MDI

وهو البديل الافتراضى حسب النظام، إلى Child.pas، وإختر زر الـ Save.

٦- إختر الـ ProjectOptions لفتح الـ Options dialog. قم بإبراز الـ ChildForm المذكور فى اللوح الإيسر، وإضغط زر السهم الأيمن الأحادى الخط لنقل الـ object من قائمة الـ Auto-create forms إلى قائمة الـ Available forms إذا لم تقم بهذه الخطوات، يتم إنشاء Child window بصورة تلقائية عندما يبدأ البرنامج.

لا يوجد شئ خطأ فى هذا من الناحية الفنية، ولكن النتيجة قد تحير المستخدمين الذين يتوقعون فتح وإنشاء نوافذ child باستخدام أوامر مثل File/New أو File/Open. تقوم معظم تطبيقات الـ MDI بإنشاء نوافذها الـ child تحت تحكم البرنامج، وليس بصورة تلقائية عند تشغيل البرنامج، أغلق الـ Project Options dialog.

إن الخطوات السابقة تنشئ غلاف الـ MDI الهيكلى. يمكنك الآن إدخال البرمجة فى الغلاف لإقامة نوافذ child إستجابة لأوامر مثل الـ File/New والـ File/Open. يوضح الفصل التالى كيفية كتابة هذا الـ code.

**فكرة:** إن النوافذ الـ child لتطبيق الـ MDI لا يجب أن تكون كلها من نفس النوع. إذا كان تطبيقك يحتاج أنواع مختلفة من النوافذ الـ child، إضافة أى عدد تحتاجه من الـ form الإضافية وذلك بتكرار الخطوات من ٣ إلى ٥. قم بإنشاء Name فريد لكل form جديدة، وحدده بـ fsMDIChild.

إحفظ المشروع باستخدام أمر الـ File/Save All وإدخل اسماً فريداً لل ملف الـ pas. الخاص بالـ unit على سبيل المثال، يمكنك أن تسمى نوافذك الصغيرة بـ Child1Form وChild2Form وتحفظها جميعاً كملفات تسمى Child1.pas وChild2.pas. لمزيد من المعلومات عن إنشاء أنواع متعددة من الـ forms النافذة الصغيرة، أنظر «العمل مع نوافذ صغيرة مختلفة النوع» فى هذا الباب.

فى هذه المرحلة، إذا كنت متابعاً، فمن بين المعلومات الأخرى فى دليل مشروعك يمكنك أن تجد الـ Pascal modules الثلاث هذه:

\* Child.pas - هذه هي ال Child form unit module . تحتوي ال module على برمجة مخصصة لنوع الوثيقة أو معلومات أخرى متوفرة بواسطة Child window . وقد يكون أيضاً للنافذة الصغيرة قائمة ، والتي يتم دمجها في العادى فى قائمة ال form الرئيسية .

\* Main.pas هذه هي ال Child form unit module ، أدخل ال event handlers فى هذه ال unit لأية object فى ال form الرئيسية ، وكذلك لبنود القائمة . ومن الناحية النموذجية ، فهناك واحد على الأقل يجب أن ينشئ أمثلة لل Child window للبرنامج . لمزيد من التفاصيل عن هذا ال code ، أنظر «Child window» و «إنشاء حالات Child window» لاحقاً فى هذا الباب .

\* MDITest.dpr - هذا هو ملف مشروع البرنامج . أنه نادراً ما يحتاج إلى أية تعديلات لتطبيقات ال MDI .

تلمذة: أصنع نسخة من دليل مشروعك الآن وإستخدم مشروع ال MDITest.dpr لتجربة تقنيات البرمجة المتنوعة الموجودة فى هذا الباب .

### إنشاء قائمة ال MDI الرئيسية

إن كل تطبيق MDI يجب أن يكون له قائمة رئيسية . وهناك أمراً واحداً على الأقل فى هذه القائمة يجب أن ينشئ Child form object - وهذا هو أمر ال FileNew . إتبع الخطوات التالية لإنشاء القائمة لتطبيق ال MDITest الذى قمت بإنشائه فى الفصول السابقة . أفتح مشروع ال MDITest.dpr ، إذا لزم الأمر ، ثم إتبع هذه الخطوات :

١- أضف MainMenu من لوحة ال Standard فى ال MainForm . (تأكد من إختيار ال MainForm ، وليس ال ChildForm) . إضغط مرتين ال MainForm لفتح ال Menu Designer الخاص بـ Delphi .

٢- أكتب File& (علامة ال & تشير إلى أن الحرف التالى ، F ، هو مفتاح القائمة) إضغط Enter ويقوم ال Menu Designer بإنشاء قائمة File . إضغط File فى Menu Designer - هذا يختار ال TMenuItem (File1) فى ال Object Inspector . قم بتغيير خاصية ال Name لل File1 لتصبح FileMenu .

## الباب الخامس عشر : تطوير تطبيقات الـ MDI

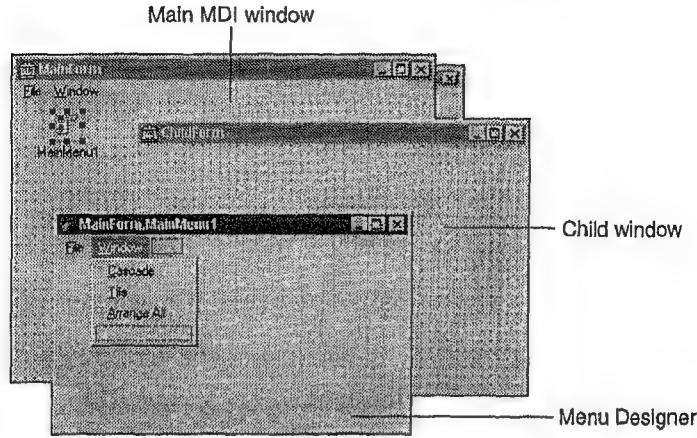
٣- مازلنا نستخدم الـ Menu Designer ، أضف أوامر الـ &New ، &Open... ، و...Save & لقائمة الـ File . لتفعل هذا ، اضغط تحت عنوان قائمة الـ File وأكتب الأوامر . كل أمر يتم إنشاؤه على أنه TMenuItem .

وفي هذا الباب ، لقد استخدمت أسماء الـ object الافتراضية New1 ، Open1 ، و Save1 ، ولكن يمكن إختيار كل object بإستخدام الـ Menu Designer وتغيير خصائص الـ Name لهم في نافذة الـ Object Inspector . للقائمة الآن أوامر File|New ، File|Open... ، و File|Save... وهو الحد الأدنى المطلوب من جانب تطبيقات الـ MDI . (إن الفصل «إنشاء Child window في هذا الباب يشرح كيفية كتابة code لإنشاء Child window ، وفتح وحفظ الوثائق ، عندما يختار المستخدم أوامر القائمة هذه) .

٤- مازلنا نستخدم الـ Menu Designer ، قم بإنشاء قائمة ثانية تسمى Windows . لتفعل هذا ، اضغط بعد الـ File وأكتب &Window . اختر قائمة الـ Window (إضغطها في Menu Designer) .

وإستخدام نافذة الـ Object Inspector ، قم بتغيير خاصية الـ Name للـ TMenuItem هذا من Window1 ، وهو البديل الافتراضي له ، إلى WindowMenu .

٥- نعود مرة أخرى إلى الـ Menu Designer ، اضغط تحت الـ Window ، وإدخل ثلاثة أوامر لهذه القائمة : &Tile ، &Cascade ، و &Arrange All . كما هو الحال في أوامر قائمة : &Tile ، &Cascade ، و &Arrange All . كما هو الحال في أوامر قائمة الـ File يمكنك إختيار كلاً من object الأوامر هذه وتغيير خصائص الـ Name لها ، ولكن لبرنامج العرض هذا ، إننا نستخدم أسماء object طبقاً للنظام الافتراضية لـ Delphi وهي Cascade1 ، Tile1 ، و Arrange1 . لتعرف كيف تكتب event handler لهذه الأوامر وأوامر أخرى لقائمة الـ Window ، أنظر فصل «إستخدام أوامر قائمة Window» في هذا الباب . يوضح شكل ١٥-١ الـ Menu Designer وقائمة الـ MDI التي تتطور في هذه المرحلة .



شكل ١٥-١ Menu Designer الخاص بـ Delphi مع عناصر قائمة مقترحة لتطبيق الـ MDI.

٦- أغلق الـ Menu Designer أو إدفعه جانباً، وإختر الـ MainForm (إضغط F12 أو استخدم الـ View/Project Manager لتجد الـ MainForm). استخدم الـ Object Inspector لتحديد خاصية الـ WindowMenu للـ MainForm بـ WindowMenu و TMenuItem. لتفعل هذا إضغط السهم الواقع بعد خاصية الـ WindowMenu وإختر الـ WindowMenu من القائمة الناتجة للـ object المتاحة للـ form. هذا التحديد يجعل التطبيق يذكر بصورة تلقائية عناوين الـ Child window المفتوحة في القائمة المستهدفة - لا يجب عليك كتابة أية code لتجعل هذا يحدث.

بالطبع، إن أوامر القائمة الدقيقة في تطبيقك تعتمد على إحتياجات برنامجك. ولكن في الحد الأدنى، يجب أن يكون لتطبيق الـ MDI أوامر لإنشاء وفتح الـ Child window لوثيقة، وقائمة، وغالباً ما تسمى Window، لذكر عناوين النوافذ المفتوحة.

**تحذير:** يمكنك التعيين لخاصية الـ WindowMenu الـ top-level menu object فقط - بعبارة أخرى، TMenuItem الذي يمثل عنصر قائمة في menu bar النافذة الرئيسية. لا تقم بتعيين أمر قائمة مثل الـ Open1 أو الـ Save1 لهذه الخاصية.



## الباب الخامس عشر : تطوير تطبيقات الـ MDI

إن Child window form يمكن أيضاً أن يكون لها MainMenu . عندما يقوم المستخدم بتنشيط حالة للـ Child window ، تقوم أوامر القائمة الخاصة بها بالإندماج بصورة تلقائية في قائمة form أساسية طبقاً لخاصية الـ GroupIndex الخاصة بعناصر القائمة أنظر «القوائم المدمجة» في هذا الباب لتعرف معلومات عن دمج قوائم الـ MDI .

### الوصول للـ Child window

تقوم ثلاث خصائص للـ form بتوفير الوصول للـ Child window ، وإنك تستخدمها في أغلب تطبيقات الـ MDI . وخلال هذا الباب العديد من الأمثلة على كل من هذه الخصائص الثلاث :

● **ActiveMDIChild** - تعتبر إلى الـ Child window النشطة حالياً . إذا لم يوجد الـ Child window ، تكون الـ ActiveMDIChild مساوية لـ nil . هذه الخاصية تعتبر مرجع للـ TForm ، ولذلك ، فإنك تحتاجها عادة لوضع الـ ActiveMDIChild في الـ Child window class مثل الـ TChildForm .

● **MDIChildCount** - العدد الصحيح للـ Child window التي تمتلكها النوافذ الأم للـ MDI . إذا لم يوجد Child window ، فإن الـ MDIChildCount تساوى صفر .

● **MDIChildren** - array للـ TForm لكل الـ Child window المملوكة لنافذة الـ MDI الرئيسية . والتعبير [0]MDIChildren يرجع إلى النافذة الصغيرة الأولى في الـ array ؛ والتعبير [1 - MDIChildCount]MDIChildren يعود على آخر Child window .

إذا كان الـ ActiveMDIChild يساوى nil ، أو إذا كان الـ MDIChildCount يساوى صفر ، فلا توجد Child window . في مثل هذه الحالات ، لا يجب أن يشير البرنامج إلى الـ MDIChildren array ، بالرغم من أن فعل هذا لا يؤدي بالضرورة إلى توليد exception . وبالتحديد ، يكون التعبير [0]MDIChildren مساوياً لـ nil عندما تكون الـ MDIChildCount تساوى صفرأ .



## الـ Child window:

إن الـ Child window فى تطبيق الـ MDI تعمل بطريقة مشابهة للنافذة الرئيسية فى تطبيق أحادى النافذة. الـ Child window يمكن أن يكون component objects مثل الـ Memos والـ Buttons. وتعرض الـ Child window وثيقة - على سبيل المثال، ملف نص أو bitmap. يمكن للمستخدمين أن يفتحوا 1 Child window متعددة، وترتب فى نظام متوال، وتصغر إلى أيقونات.

للـ Child window يمكن أن يكون toolbars و Status line، ولكن من الناحية التقليدية، مثل هذه الأنواع من الـ object تظهر عادة فى النافذة الرئيسية.

إن إختيار الـ Child window يمكن أيضاً أن يدمج أوامر فى القائمة الرئيسية للتطبيق. وهذه التقنية مفيدة بالأخص عندما تستخدم أنواعاً متعددة من الـ Child window، كلاً بمتطلبات الأوامر الخاصة بها. يوضح الفصل التالى تقنيات البرمجة لهذه الـ Child window ويذكر غلاف تطبيق الـ MDI الذى يمكنك استخدامه لتبدأ برنامجاً جديداً.

## الـ Child window ذات النوع واحد:

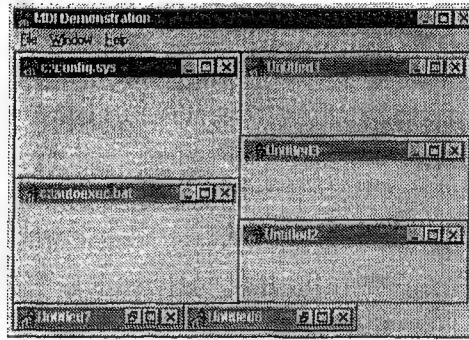
**على القرص المدمج:** إن تطبيق الـ MDIDemo على القرص المدمج فى دليل الـ Source\MDIDemo يوضح كيفية إنشاء الـ Child window ذات نوع واحد.



يوضح البرنامج كيفية برمجة أوامر القائمة العامة مثل الـ File|New، الـ File|Open...، الـ Window|Cascade والـ Window|Tile. يوضح شكل ١٥-٢ عرض البرنامج مع بعض النوافذ المفتوحة.

توضح القائمة ١٥- module الـ Child.pas لتطبيق الـ MDIDemo. غنى هذا البرنامج، لا تعرض النافذة الصغيرة بيانات حقيقية، لذا فإن الـ module تكون بسيطة. ولكنها، توضح أدنى حد من الـ procedures يمكن أن توفرها الـ Child window.

## الباب الخامس عشر : تطوير تطبيقات الـ MDI



شكل ١٥-٢: يوضح تطبيق الـ MDIDemo

برمجة Child window ذات نفس النوع.

القائمة ١٥-١: MDIDemo\Child.pas

```
unit Child;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics,  
Controls,
```

```
Forms, Dialogs, StdCtrls;
```

```
type
```

```
TChildForm = class(TForm)
```

```
    procedure FormClose(Sender: TObject;
```

```
        var Action: TCloseAction);
```

```
    private
```

```
    { Private declarations }
```

```
    public
```

```
    { Public declarations }
```

```
    procedure LoadData(const FileName: String); virtual;
```

```
    procedure SaveData(const FileName:  
        String); virtual;
```

```
end;
```

```

var
    ChildForm: TChildForm;

implementation

{$R *.DFM}

procedure TChildForm.FormClose(Sender: TObject;
    var Action: TCloseAction);
begin
    Action := caFree;
end;

procedure TChildForm.LoadData(const FileName: String);
begin
    ShowMessage('LoadData from ' + FileName);
    Caption := LowerCase(FileName);
end;

procedure TChildForm.SaveData(const FileName: String);
begin
    ShowMessage('SaveData to ' + FileName);
    Caption := LowerCase(FileName);
end;

end.

```

إن الـ Child window module يمكن أن توفر procedure لتحميل وحفظ بيانات وثيقة. إن procedures مثل الـ LoadData والـ SaveData يتم تعريفها كأعضاء عامة من الـ TChildForm class. إن السبب في جعلها عامة هو أن أوامر الـ Save والـ Open للـ modules الرئيسية يمكن أن تستدعيها.

وتقرر القائمة أيضاً الـ LoadData والـ SaveData الفعلية. وكنتيجه، يمكن أن ترث module أخرى الـ TChildForm class وتغطي على الـ procedure. على سبيل المثال، يمكنك إضافة Child form للبرنامج وتأسيس الـ form class

## الباب الخامس عشر : تطوير تطبيقات ال MDI

على ال TChildForm ، بدلاً من ال TForm المعتادة . يمكنك عندئذ كتابة ال LoadData procedures وال SaveData لقراءة وكتابة البيانات الخاصة لل module . إن وراثه class بهذه الطريقة غالباً ما يسمى subclassing a form (أو ال class الفرعية ل form) ، وهى تقنية موضحة فيما بعد فى هذا الباب .

**Tip** **فكرة:** يجب عليك دائماً أن توفر OnClose لكل من ال Child window . حدد ال Action parameter بـ caFree حتى يترك التطبيق ال Child window عندما تغلق نافذته . إذا لم تحدد ال Action بـ caFree ، فإن إغلاق النافذة يؤدى إلى تصغير حجمها فى ال Child window client area بالنافذة الرئيسة .

بالإضافة إلى ال OnClose ، يمكنك أيضاً أن تكتب OnCloseQuery لتحذر المستخدمين عندما يغلقوا نافذة بيانات غير محفوظة . فى ال procedure ، حدد ال OnClose بـ False لتمنع النافذة من الإغلاق ، حدد ال CanClose بـ True إذا كان من الممكن إغلاق النافذة . انظر فصل "إغلاق نافذة ٨" فى الباب الثالث . لمزيد من المعلومات عن ال OnClose وال OnCloseQuery . إن النوافذ الرئيسة لتطبيق ال MDI لا يمكن أن تغلق ، وبذلك ، لا يمكن أن ينتهى التطبيق ، إلا إذا كان من الممكن إغلاق جميع ال Child window . وهذه الميزة الرائعة تساعد على منع فقد البيانات فى ال Child window مخفية وراء نافذة أخرى .

إن ال LoadData procedures وال SaveData لل MDIDemo لا تقرأ أو تكتب أية ملفات حقيقية ، لذا فلك الحرية فى أن تلعب بأوامر البرنامج Open... ، Save ، و Save As . للتأكيد أن البرنامج يستدعى ال procedures فى الأوقات السليمة ، تعرض ال ShowMessage أسم الملف ، تقوم LoadData procedures وال SaveData بتعيين اسم الملف الحالى لل Caption الخاص بـ ال Child form . وهذا يعرض اسم الملف ومساره فى ال address bar النافذة .

يمكنك ببساطة تعديل ال MDIDemo ليقرأ ويكتب بيانات حقيقية . على سبيل المثال ، أدخل Memo فى ال ChildForm . (إذا لم تكن ترى هذه النافذة افتح ال MDIDemo.dpr ، اختر أمر ال ViewProject Manager ، اضغط علامة الزائد الواقعة بعد ال Child ، واضغط مرتين ال ChildForm) .

حدد خاصية الـ Align للـ Memo1 الجديد بـ alClient، والتي تجعل الـ object ميلاً نافذته تماماً. اختر خاصية Font مناسبة إذا أردت، احذف الـ "Memo1" من خاصية الـ Lines (اضغط الزر البيضاوي لفتح الـ editor)، وحدد الـ ScrollBars بـ ssBoth.

لإعادة برمجة الـ ChildForm module، باستخدام الـ Project Manager، افتح ملف الـ Child.pas واختره في الـ code editor. وفي مكان عبارات الـ ShowMessage من القائمة (١٥-١)، ادخال الأوامر التالية لقراءة وكتابة ملفات النص.

```
{ Replace ShowMessage statement in LoadData with: }
Memo1.Lines.LoadFromFile(FileName);
```

```
{ Replace ShowMessage statement in SaveData with: }
Memo1.Lines.SaveToFile(FileName);
```

You have just constructed an MDI text editor!

لقد أنشأت الآن محور نص الـ MDI. اضغط F9 لتشغيل البرنامج المعدل. اختر أوامر الـ File|New لفتح الـ Child window، والتي يمكنك أن تكتب داخلها نصاً أو تقوم بتغييرات. استخدم الـ File|Save لحفظ تغييراتك على قرص.

**تحذير:** ان الـ code السابق تجعل الـ MDIDemo تطبيقاً "حياً". والتغييرات التي تصنعها بالملف وتضغطها على القرص تكون دائمة. اذا كنت تلهو فقط، فاحفظ نسخ من أى ملفات تحررها.



ان اسماء المسارات الطويلة قد تبدو فوضوية في الـ Captions الخاصة بالنوافذ، وقد تريد ان تستخدم الـ ExtractFileName function لتحسين اسماء ملف لـ address Bar بين الـ Child window والـ drive المسار والـ drive. لتفعل هذا، اضع الـ SysUtils إلى عبارة الـ Uses للـ module، وعرف متغير الـ String لتستخدمه في عبارات المدخلات والمخرجات ولتعيينه لخاصية الـ Caption للنافذة نتيجة الـ ExtractFileName function من الـ SysUtils unit. انظر "نظم إدارة الملف" في الـ Online help لـ Delphi لمعرفة المزيد من المعلومات عنها وعن الـ file function أخرى.

## إنشاء Child window Instance

على القرص المدمج : توضح القائمة ١٥-٢ تعريف الـ TMainForm class للـ MDIDemo وOnClick لقائمة الـ File . وهذه الجزئية من القائمة توضح كيفية إنشاء نماذج جديدة من الـ TChildForm class . ولأن الـ module الرئيسية للـ MDIDemo طويلة، فلذلك قمت بتقديمها في أجزاء . لفحص القائمة الكاملة، قم بتحميل مشروع الـ MDIDemo.dpr في Delphi من على القرص المدمج في دليل الـ Source\MDIDemo .



القائمة ١٥-٢: OnClick لقائمة File للـ MDIDemo .

```
unit Main;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics,  
Controls,  
Forms, Dialogs, Menus, Child;
```

```
type
```

```
TMainForm = class(TForm)  
  MainMenu1: TMainMenu;  
    FileMenu: TMenuItem;  
    FileOpen: TMenuItem;  
    FileSave: TMenuItem;  
    FileSaveAs: TMenuItem;  
    FileNew: TMenuItem;  
    N1: TMenuItem;  
    FileExit: TMenuItem;  
    WindowMenu: TMenuItem;  
    WindowCascade: TMenuItem;  
    WindowTile: TMenuItem;  
    WindowArrangeIcons: TMenuItem;  
    N2: TMenuItem;
```

```

WindowCloseAll: TMenuItem;
WindowMinimizeAll: TMenuItem;
HelpMenu: TMenuItem;
HelpAbout: TMenuItem;
OpenDialog: TOpenDialog;
FileClose: TMenuItem;
N3: TMenuItem;
SaveDialog: TSaveDialog;
procedure FileNewClick(Sender: TObject);
procedure FileOpenClick(Sender: TObject);
procedure FileCloseClick(Sender: TObject);
procedure FileSaveClick(Sender: TObject);
procedure FileSaveAsClick(Sender: TObject);
procedure FileExitClick(Sender: TObject);
procedure WindowCascadeClick(Sender:
TObject);
procedure WindowTileClick(Sender: TObject);
procedure WindowArrangeIconsClick(Sender:
TObject);
procedure WindowMinimizeAllClick(Sender:
TObject);
procedure WindowCloseAllClick(Sender:
TObject);
procedure HelpAboutClick(Sender: TObject);
procedure FileMenuClick(Sender: TObject);
procedure WindowMenuClick(Sender: TObject);
private
{- Private declarations }
procedure CreateChild(const Name: string);
public
{- Public declarations }
end;
var
MainForm: TMainForm;

implementation

```

=====

{ \$R \*.DFM }

```
const
    maxChildren = 10; { Optional: No maximum required }
```

```
procedure TMainForm.CreateChild(const Name: String);
var
    Child: TChildForm;
begin
    Child := TChildForm.Create(Application);
    Child.Caption := Name;
end;
```

```
procedure TMainForm.FileNewClick(Sender: TObject);
begin
    CreateChild('Untitled' + IntToStr(MDIChildCount + 1));
end;
```

```
procedure TMainForm.FileOpenClick(Sender: TObject);
begin
    if OpenFileDialog.Execute then
    begin
        CreateChild(Lowercase(OpenDialog.FileName));
        with ActiveMDIChild as TChildForm do
            LoadData(OpenDialog.FileName);
    end;
end;
```

```
procedure TMainForm.FileCloseClick(Sender: TObject);
begin
    if ActiveMDIChild <> nil then
        ActiveMDIChild.Close;
end;
```

```
procedure TMainForm.FileSaveClick(Sender: TObject);
begin
```



```

if Pos('Untitled', ActiveMDIChild.Caption) = 1 then
    FileSaveAsClick(Sender)
    else with ActiveMDIChild as TChildForm do
        SaveData(Caption);
end;

procedure TMainForm.FileSaveAsClick(Sender: TObject);
var
    FExt: String;
begin
    with SaveDialog do
        begin
            FileName := ActiveMDIChild.Caption;
            FExt := ExtractFileExt(FileName);
            if Length(FExt) = 0 then
                FExt := '.*';
            Filter := 'Files (*' + FExt + ')*|*' + FExt;
            if Execute then
                with ActiveMDIChild as TChildForm do
                    SaveData(FileName);
                end;
        end;
end;

procedure TMainForm.FileExitClick(Sender: TObject);
begin
    Close;
end;

```

إن الـ CreateChild procedure، والمقرر كعضو خاص في الـ TMainForm class، ينشئ instances من الـ TChildForm. والبنود المقرر كعناصر خاصة تكون قابلة للوصول إليها فقط من الـ methods في الـ class. هذا يساعد على ضمان أن الـ modules الأخرى لا تقوم باستدعاء الـ procedures والـ functions والتي تؤدي خدمات هامة بلا تمييز. ويوضع الـ CreateChild procedure أيضاً كيفية إنشاء form في وقت التشغيل. لتفعل هذا، قم بتعريف متغير من نوع الـ class:

## الباب الخامس عشر : تطوير تطبيقات ال MDI

var

Child: TChildForm;

بعد ذلك ، وفي هيكل ال procedure ، قم باستدعاء ال Create methods لل class مع ال Application ك argument قم بتعيين ال object الناتج إلى المتغير :

Child := TChildForm.Create(Application);

إن السبب في تمرير ال Application ، وليس ال MainForm ، إلى ال Create هو أن ال Application يمثل نافذة الإطار لل MDI ، وهى النافذة الحقيقية التى تملك ال Child window . وال MainForm ما هى إلا أمر بديلة لصغارها .

إن FileNewClick procedure هو ، ال FileNew event handler ، يستدعى ال CreateChild . و FileOpen يقوم بنفس الشئ ، ولكن أيضاً يستدعى ال LoadData procedure لل Child.pas . ولكن ، وما هذه إلا تقنيات مقترحة لإقامة ال Child window وتحميل بيانات من على ملفات ؛ فأنت حر فى كتابة ال module كما تشاء .

إستخدم خاصية ال ActiveMDIChild لل MainForm لأداء عمليات على ال Child window الحالية . وهذه الخاصية تساوى nil إذا لم يوجد ال Child window ، ويجب أن تفحص هذا دائماً قبل أن تستخدم الخاصية . على سبيل المثال ، إن ال FileClose event وهى ال FileCloseClick procedure ، تغلق ال Child window النشطة بعبارة :

if ActiveMDIChild <> nil then

ActiveMDIChild.Close;

لا تستخدم أبداً عبارة غير مقيدة مثل التالية ، التى تولد exception إذا لم يوجد ال Child window .

هذا يحدث لأن العبارة قد تحاول أن تستدعى ال method (Close) ك

: nil reference

ActiveMDIChild.Close; /// ???

كما إقترحت، يجب أن نتأكد من أن ال Child window module تحفظ البيانات الخاصة بها في OnClose و OnCloseQuery. لا تغلق النافذة فعلياً إلا إذا كانت ال Child object تحققت من أنها يمكنها أن تغلق.

إن ال FileSaveClick procedures وال FileSaveAsClick لل MainForm تقوم بإستدعاء ال SaveData procedure لل Child window النشط. إذا كان Caption النافذة Untitled، فإن ال FileSaveClick هو الافتراضى لـ File save-as click؛ وإلا، تقوم ال code بإستدعاء ال SaveData مع اسم الملف الحالى، والمأخوذ من Caption النافذة. ويستخدم ال FileSaveAsClick التقنية المقترحة فى الباب الثانى عشر لقصر ملفات ال SaveDialog على أولئك الذين لهم نفس الإمتداد كإمتداد النافذة الحالية. وأخيراً، فى هذه الجزئية من القائمة، إن ال FileExit event handler لل MainForm، FileExitClick، يستدعى ال Close.

إن بإمكانه أن يفعل هذا لأن ال object الأم يحاول أن يغلق ويحرر كل ال Child window. ولكن، ينتهى التطبيق فقط إذا كان من الممكن إغلاق كل ال Child window.

### إستخدام أوامر قائمة ال Window

إن أغلب تطبيقات ال MDI توفر قائمة Window، والتى تؤدى عمليات على ال Child window مثل التتالى، التجانس، والترتيب للأيقونات فى ال client area بالنافذة الرئيسية. توضح القائمة ١٥-٣ OnClick لقائمة ال Window لل MDIDemo وتظهر كيفية كتابة أوامر نافذة مخصصة مثل ال Close، التى لا يقدمه ال Delphi.

#### القائمة ١٥-٣: OnClick لقائمة ال Window لل MDIDemo.

```
procedure TMainForm.WindowCascadeClick(Sender: TObject);
begin
  Cascade;
end;
```

```
procedure TMainForm.WindowTileClick(Sender: TObject);
```

## الباب الخامس عشر : تطوير تطبيقات الـ MDI

```
begin
    Tile;
end;

procedure TMainForm.WindowArrangeIconsClick(Sender: TObject);
begin
    ArrangeIcons;
end;

procedure TMainForm.WindowMinimizeAllClick(Sender:
TObject);
var
    I: Integer;
begin
    for I := MDIChildCount - 1 downto 0 do
        MDIChildren[I].WindowState := wsMinimized;
end;

procedure TMainForm.WindowCloseAllClick(Sender: TObject);
var
    I: Integer;
begin
    for I := 0 to MDIChildCount - 1 do
        MDIChildren[I].Close;
end;
```

إن الـ procedures الثلاثة الأولى تنفذ أوامر قائمة الـ Window المعيارية : Arrange ، Tile ، Cascade الأيقونات . ولأن هذه الأوامر عامة جداً ، فإن الـ TForm class توفر الـ code اللازمة في الـ Cascade method والـ Tile والـ ArrangeIcons كإستجابة لأوامر القائمة المناسبة . يمكنك أيضاً إستدعاء هذه الـ method في أوقات أخرى - على سبيل المثال ، إستجابة للـ OnClick الخاص بالـ SpeedButton في الـ toolbar .

هناك أيضاً اثنين من الأوامر المعيارية التي يمكنك أن تضيفها لقائمة الـ Window هما الـ Next والـ Previous . ببساطة إستدع الـ Previous methods

وال Next لل form الرئيسية . ولكن ، قد تريد إبطال هذه الأوامر إذا كانت نافذة واحدة فقط تستخدم عبارات مثل التالية فى `OnClick` الخاص بقائمة ال `Window` (بفرض ال `WindowNext` وال `WindowPrevious` ، `TMenuItem objects` )

`WindowNext.Enabled := MDIChildCount > 1;`

`WindowPrevious.Enabled := WindowNext.Enabled;`

بالإضافة لك وأمر المعيارية لقائمة ال `Window` يمكنك كتابة أوامر جديدة لأداء عمليات على كل ال `Child window` . على سبيل المثال ، يقوم ال `MDIDemo` بتنفيذ أمر ال `Window!Minimize all` مع ال `for loop` هذه :

`for I := MDIChildCount - 1 downto 0 do`

`MDIChildren[I].WindowState := wsMinimized;`

لتجعل الأيقونة تخرج فى ترتيب معقول ، تقوم ال `for loop` بالعد من آخر `Child window` إلى الأولى . إستخدم `array` ، `MDIChildren` لل form النافذة الرئيسية للوصول إلى كل `Child window` . وعناصر ال `array` من نوع ال `TForm` ، لذا فإن لم تكن تستخدم خاصية أو `method` مورث مثل ال `WindowState` الموضح هذا ، فإنك تحتاج أن تكتب تعبيراً لإستدعاء `methods` وإستخدام خصائص ال `Child window class` الخاصة بك . على سبيل المثال ، العبارات التالية تستدعى ال `YourMethod procedure` لل `Child window` الأولى :

`if MDIChildCount > 0 then`

`TChildForm(MDIChildren[0]).YourMethod;`

وبالتبادل إستخدم عبارة `with` مثل هذه :

`if MDIChildCount > 0 then`

`with MDIChildren[0] as TChildForm do`

`YourMethod;`

إن تطبيق ال `MDI` المصمم بشكل سليم يجب أن يبحث دائماً عن ال `Child window` واحدة على الأقل قبل أداء أعمال من خلال ال `MDIChildren` . أما أن يبحث ما إذا كان ال `MDIChildCount` أكبر من صفر ، أو يختبر ما إذا كان ال `ActiveMDIChild` لا يساوى `nil` .

## الباب الخامس عشر : تطوير تطبيقات الـ MDI

بالرغم من أن ليس كل تطبيقات الـ MDI توفر أمر الـ Window|Close، ربما كان يجب أن يكونوا كذلك. إنني أحاول أن أفتح العديد من النوافذ وأنا أعمل على مختلف المشروعات ومن المفيد أن يكون لديك method لإغلاق جميع النوافذ دون أن تضطر إلى إغلاقها واحدة واحدة. يؤدي الـ MIDemo هذا العمل في الـ procedure، WindowCloseAllClick، والذي ينفذ الـ loop لإغلاق النوافذ بالترتيب من الأولى إلى الأخيرة:

```
for I := MDIChildCount - 1 downto 0 do
```

```
MDIChildren[I].Close;
```

داخلياً، يتم تمثيل الـ MDIChildren كـ TList object، ولذلك، فمن المسموح أن تؤدي عمليات مثل السابقة التي تؤثر على عدد الـ Child window.

### مجموعة من أوامر الـ MDI

توضح القائمة ١٥-٤ باقي الـ source code، Main.pas للـ MDIDemo.

القائمة ١٥-٤: procedures متنوعة للـ MDIDemo.

```
procedure TMainForm.HelpAboutClick(Sender: TObject);
```

```
begin
```

```
    AboutForm.ShowModal;
```

```
end;
```

```
procedure TMainForm.FileMenuClick(Sender: TObject);
```

```
begin
```

```
    FileNew.Enabled := MDIChildCount < maxChildren;
```

```
    FileOpen.Enabled := FileNew.Enabled;
```

```
    FileClose.Enabled := MDIChildCount > 0;
```

```
    FileSave.Enabled := FileClose.Enabled;
```

```
    FileSaveAs.Enabled := FileClose.Enabled;
```

```
end;
```

```
procedure TMainForm.WindowMenuClick(Sender: TObject);
```

```
var
```

```

I: Integer;
begin
  with WindowMenu do
    for I := 0 to Count - 1 do
      with Items[I] as TMenuItem do
        Enabled := MDIChildCount > 0;
      end;
    end.
  end.

```

إن الـ `HelpAboutClick` procedure يعرض الـ `AboutBox` dialog الخاص بالبرنامج، وهو غير موضح هنا. إن الـ `FileMenuClick` Procedures والـ `WindowMenuClick` هي الـ `OnClick` للـ `FileMenu` والـ `WindowMenu`. يستدعي البرنامج هذه الـ `procedures` عندما يفتح المستخدم القوائم تقوم الـ `procedures` بإبطال وتشغيل الأوامر على أساس ظروف البرنامج. على سبيل المثال يتم إبطال الـ `FileNew` عندما يكون الـ `MDIChildCount` مساوياً لـ `maxChildren`، والذي يحد من عدد الـ `Child window` التي يفتحها المستخدم. إن كل أوامر قائمة الـ `Window` يتم إبطالها عندما لا يوجد `Child window`. إذا أضفت أوامر `Next` و `Previous` كما هو مقترح، يجب أن تقوم بتشغيلها منفصلة إذا كان الـ `MDIChildCount` أكبر من واحد.

**ملحوظة:** إنك لا تحتاج أن تحد من عدد الـ `Child window`. لقد ضمنت هذه الميزة في الـ `MDIDemo` فقط لإظهار البرمجة. يمكن لتطبيق الـ `MDI` أن يكون له أي عدد من الـ `Child window` بقدر ما تسمح الذاكرة والـ `Resource` الأخرى.

Note

## العمل مع الـ `Child window` مختلفة النوع

**على القرص المدمج:** إن الـ `Child window` لتطبيق الـ `MDI` قد تكون من أنواع مختلفة. لكل نوع من الـ `Child window`، أضف `form` جديدة للمشروع وأضف برمجة للـ `module` الرئيسية لإنشاء `instances` نافذة. على سبيل المثال، قم بتجربة الخطوات التالية لتشغيل الـ



## الباب الخامس عشر : تطوير تطبيقات الـ MDI

Child window MDIDemo تستطيع عرض ملفات bitmap. (يوجد البرنامج التام على القرص المدمج فى دليل Source\MDIDemo2. إذا لم تكن تريد القيام بهذه التعديلات بنفسك، أفتح ملف مشروع MDIDemo2.dpr فى هذا الدليل لترى الملفات التامة مع Delphi. هذه الخطوات تفترض أنك تقوم بتعديل المشروع الأصيلى فى الـ Source\MDIDemo).

١- أفتح ملف مشروع MDIDemo.dpr. أضف module أخرى للـ MDIDemo بإختيار الـ File\New Form أو بضغط الـ New Form SpeedButton.

٢- قم بتغيير خاصية الـ Name للـ form الجديدة Form1 إلى ChildBmpForm. حدد الـ FormStyle بـ fsMDIChild.

أختر Project\Options وأنقل الـ ChildBmpForm من قائمة الـ Auto-create forms إلى قائمة الـ Available forms يجب أن تذكر أن الـ panel اليسرى الـ MainForm والـ AboutForm. أغلق الـ Project Options dialog بضغط OK.

٣- لحفظ المشروع بإختيار الـ File\Save All. عندما يطلب منك اسم ملف، أدخل Childbmp.pas بدلاً من الاسم الافتراضى، Unit1.pas.

٤- أضف الـ ChildBmp form بالضغط داخل نافذتها. إنتقل إلى الـ Object Inspector، إضغط Events، وإضغط مرتين المسافة الواقعة إلى اليمين من الـ OnClose لإنشاء event handler لهذا الـ event. أكتب عبارة تعين الـ caFree للـ Action. وهذا يحو الـ form من الذاكرة عندما تغلق الـ Child window. إذا لم تقم بهذه الخطوة فإن إغلاق النافذة يؤدي إلى تصغيرها فى الـ client area للنافذة الرئيسية - هذا مسموح، ولكنه فى الغالب ليس هو ما تريد. يجب أن يبدو OnClose مثل هذا:

```
procedure TChildBmpForm.FormClose(Sender: TObject;  
  var Action: TCloseAction);  
begin  
  Action := caFree;  
end;
```



٥- أضف Image من Additional palette فى ال ChildBmp form .  
حدد خاصية ال Align لل Image1 بـ alClient حتى تملأ النافذة وحدد خاصية ال  
Image1 Stretch بـ Ture . هذه التغييرات تملأ ال Child window بالصورة التى  
تم تحميلها فى وقت التشغيل وتحدد حجم الصورة لتناسب مع ال client area لل  
Child window .

٦- إنتقل لل code editor وأختر ChildBmp unit . أضف Child فى  
عبارة ال uses فى أعلى ال module . هذا ضرورياً لأننا نحتاج أن تراث class من  
ال Child form الأصلية، إن عبارة ال uses التامة يجب أن تبدو مثل هذه:  
uses

Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, Child;

٧- بينما لازلنا فى ال code editor لل ChildBmp قم بتغيير ال class  
الأساسية لل TChildBmpForm من TForm إلى TChildForm هذا يجعل ال  
TChildBmpForm تراث ٨ خصائص و methods من ال TChildForm ، وهذه  
تقنية تسمى ال subclassing لل form إن تعريف ال class المعدلة لل  
TChildBmpForm يجب أن يبدو الآن مثل هذا:

type

TChildBmpForm = class(TChildForm)

...

٨- أضف ال LoadData procedures وال SaveData لل  
TChildBmpForm class . قم بتصميم ال procedures مع ال override  
directive ، الذى يخبر Delphi أن يستبدل ال methods الفعلية الموروثة من ال  
TChildForm . إدخل هذه التعاريف تحت الكلمة الرئيسية public فى تعريف  
ال class TChildBmpForm :

public

procedure LoadData(const FileName: String); override;

procedure SaveData(const FileName: String); override;

## الباب الخامس عشر : تطوير تطبيقات الـ MDI

٩- قم بتنفيذ الـ procedure فى قطاع الـ implementation للـ ChildBmp module . توضح القائمة ١٥-٥ الـ code التامة ، التى يمكنك إستخدامها كمرشد لك لإتمام برنامجك .

هذه القائمة مع القوائم الأخرى فى هذا الفصل ، موجودة فى دليل الـ Source\MDIDemo2 على القرص المدمج .

١٠- اختر الـ ChildBmpForm فى الـ Object Inspector ، إضغط الـ Events page tab ، وإضغط مرتين المسافة الواقعة إلى اليمين من الـ OnCreate event . أدخل الـ code من الـ FormCreate procedure بالقائمة ١٥-٥ . هذا يؤدى إلى تحرير الـ Memo1 الموروث والذى قد يتنازع مع الـ Image1 فى الـ class الجديدة إذا لم يتم تحريره . (القائمة ١٥-٥ والقوائم الأخرى فى هذا الفصل توجد على القرص المدمج فى دليل الـ Source\MDIDemo2 .

**القائمة ١٥-٥: الـ source code للـ ChildBmp module التامة.**

```
unit Childbmp;
```

```
interface
```

```
uses
```

```
  Windows, Messages, SysUtils, Classes, Graphics,  
  Controls,  
  Forms, Dialogs, Child, ExtCtrls;
```

```
type
```

```
  TChildBmpForm = class(TChildForm)
```

```
    Image1: TImage;
```

```
    procedure FormClose(Sender: TObject;
```

```
      var Action: TCloseAction);
```

```
    procedure FormCreate(Sender: TObject);
```

```
  private
```

```
    { Private declarations }
```

```
  public
```

```
    { Public declarations }
```

```
    procedure LoadData(const FileName: String);
```

```

        override;
        procedure SaveData(const FileName: String);
        override;
        end;

ChildBmpForm: TChildBmpForm;

implementation

{$R *.DFM}

procedure TChildBmpForm.FormClose(Sender: TObject;
    var Action: TCloseAction);
begin
    Action := caFree;
end;

procedure TChildBmpForm.LoadData(const FileName: String);
begin
    Image1.Picture.LoadFromFile(FileName);
    Caption := LowerCase(FileName);
end;

procedure TChildBmpForm.SaveData(const FileName: String);
begin
    Image1.Picture.SaveToFile(FileName);
    Caption := LowerCase(FileName);
end;

procedure TChildBmpForm.FormCreate(Sender: TObject);
begin
    inherited;
    Memo1.Free;
end;

end.

```

## الباب الخامس عشر : تطوير تطبيقات الـ MDI

**ملحوظة:** إن السبب في تحرير الـ Memo1 في الـ FormCreate method هو منع هذا الـ object من التضارب مع الـ Image1 في الـ TChildBmpForm class. هذا يظهر إحدى المشاكل الأساسية مع الـ classes الفرعية للنوافذ - قد تضيف الـ class الجديدة objects مع الـ objects في الـ class السالفة لها. انظر مشروع ١٥-٦ من المقترحات عن كيفية تحسين ميزات الـ class الخاص بالبرنامج.

Note

إن الخطوة الأخيرة في تعديل الـ MDIDemo لقراءة وكتابة ملفات bitmap هو إعادة برمجة الـ Main module بحيث تستطيع أن تنشئ حالات من الـ bitmap class الصغيرة. لتفعل هذا، اختر الـ Main module في محرر الـ code (استخدم الـ View/Project Manager لفتح الـ MainForm إذا كانت الـ Main module غير مرئية). أضف ChildBmp إلى عبارة الـ Uses الـ Main module، والتي يجب أن تكون مثل هذه:

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, Menus, Child, ChildBmp;

**على القرص المدمج:** أعد كتابة الـ CreateChild procedure كما هو موضح في القائمة ١٥-٦. (يوجد هذا النص في دليل الـ Source\MDIDemo2 على القرص المدمج.) اضغط F9 ولتشغيل البرنامج المعدل. افتح ملف bitmap. (إذا أضفت Memo إلى الـ Child module كما هو مقترح، يمكنك أيضاً فتح ملفات نص.) يقوم البرنامج بإنشاء النوع المناسب من النوافذ على أساس امتداد اسم الملف.



**القائمة ١٥-٦:** الـ CreateChild procedure الذي تم إعادة برمجته، والخاص بالـ Main module، لتطبيق الـ MDIDemo المعدل، وكذلك أضف ChildBmp لعبارة الـ Uses الـ module.

```
procedure TMainForm.CreateChild(const Name: String);
var
  Child: TChildForm;
  FExt: String;
begin
```

```

FExt := ExtractFileExt(Name);
if FExt = '.bmp' then
  Child := TChildBmpForm.Create(Application)
else
  Child := TChildForm.Create(Application);
  Child.Caption := Name;
end;

```

فى ال CreateChild procedure الجديد، إذا كان bmp . هو إمتداد الملف، ينشئ البرنامج Child window TChildBmpForm، وإن لم يكن bmp .، فإن البرنامج ينشئ TChildForm . ولأن ال LoadData وال SaveData تعتبر procedures فعلية (راجع القائمة ١٥-١ و ١٥-٥)، فإن نوع Child window object يحدد نوع البيانات التى يتم تحميلها.

وحتى تفهم جيداً قيمة استخدام methods فعلية فى ال forms من نوع ال class الفرعية، انظر ال FileOpenClick procedure الخاص بال Main module فى القائمة ١٥-٢. إذا كانت Child window هى TChildForm، يقوم ال procedure باستدعاء ال LoadData procedure الأصلى. إذا كانت النافذة هى TChildBmpForm، يقوم ال procedure باستدعاء ال method الذى تم تخطيطه. وهذه الاستدعاءات لا يتم توجيهها بواسطة ال code الظاهر فى البرنامج، ولكن بواسطة أنواع Child window المشار إليها عن طريق ال ActiveMDIChild.

وبسبب تصميم البرنامج المختص بال objects، فمن السهل نسبياً إضافة أنواع أخرى من Child window. ببساطة، قم بإنشاء وبرمجة form module جديدة كما فعلت مع ال TChildBmpForm. استخلص ال form class الجديدة من ال TChildForm واكتب ال LoadData procedures وال SaveData. قم بتعديل ال CreateChild فى ال Main module لإنشاء حالات من ال class الجديدة، وبذلك تكون قد اتممت العمل.

#### دمج القوائم،

فى تطبيقات ال MDI وبالأخص مع الأنواع المتعددة من ال Child window، قد تريد تعديل أوامر القائمة أو حتى إضافة قوائم، إعتماًداً على أى نوع من ال Child window تكون نشطة. هذا يسهل فعله، ولكن يتطلب الانتباه إلى بعض التفاصيل التى قد لا تكون واضحة.

## الباب الخامس عشر : تطوير تطبيقات الـ MDI

في النافذة الأم- تلك التي بها الـ `FormStyle` محددة بـ `fsMDIForm`- يجب أن يحتوي الـ `MainMenu` على أوامر عامة تنطبق بشكل عام على كل الـ `Child window` على سبيل المثال، توفر قائمة النافذة الأم عادة قائمة `Window` ذات أوامر `Cascade`، `Tile` وغيرها التي تؤثر على كل النوافذ بغض النظر عن نوعها. ويجب أيضاً أن يكون لقائمة النافذة الرئيسية أوامر مثل الـ `File/New`، `File/Open...`، `File/Close` التي تفتح وتغلق النوافذ الصغيرة.

في كل نافذة صغيرة تتطلب أوامر فريدة، أو تحتاج إلى تغيير بنود القائمة بناء على ظروف متنوعة، يمكنك إدخال `MainMenu` آخر. إنني أحب أن أسمى هذه الـ `objects` طبقاً للـ `module`- على سبيل المثال، `ChildFormMenu` أو `ChildBmpFormMenu`. أدخل أوامر في الـ `MainMenu` هذه للدمج مع قائمة النافذة الأم.

إن دمج القوائم في تطبيقات الـ MDI يكون تلقائياً لأي الـ `Child window` لها `MainMenu`. لهذا السبب، يجب أن تحدد كل خصائص الـ `AutoMerge` للـ `MainMenu` بـ `False`. ومن الناحية الفنية، يجب أن تكون خاصية الـ `AutoMerge` للنافذة الأم فقط هي التي تحدد بـ `False`. يتم تجاهل قيم الـ `AutoMerge` للـ `MainMenu` الـ `Child window`، ولكن حددها جميعاً بـ `False` على أي حال.

انظر الباب الخامس لمعرفة التعليمات عن دمج القوائم في تطبيقات غير الـ MDI. التطبيقات غير الـ MDI فقط هي التي تستخدم خاصية الـ `AutoMerge`.

بعد ذلك، قم بتعيين قيم `GroupIndex` إلى عناصر قائمة معينة. تحدد هذه القيم مكان إدخال القوائم وما إذا كنت سوف تستبدل عنصر قائمة أو تضيف أوامر جديدة. عند الدمج، يستخدم التطبيق قيم الـ `GroupIndex` طبقاً للقواعد التالية:

\* يتم استبدال بنود القائمة التي لها نفس قيم الـ `GroupIndex`. على سبيل المثال، إذا كان عنصر `MainMenu` له `GroupIndex` مساوياً لعشرة، فإن أي عناصر قائمة في `MainMenu` خاصة بـ `Child window` لها نفس الـ `GroupIndex` تحل محل البنود في النافذة الأم.

\* يتم إدخال عناصر القائمة ذات الـ `GroupIndex` الفريدة. قم بتعيين قيم أعلى لإدخال عناصر قائمة في الـ `Child window` إلى اليمين من البنود ذات القيم المنخفضة لإدخال عناصر قائمة إلى اليسار من العناصر في النافذة الأم.

بالإضافة إلى دمج القوائم، يمكنك أيضاً استدعاء ال TMenuItem methods لتشغيل وإبطال عناصر القائمة، لإدخال أو تغيير أوامر، ولأداء حيل أخرى للقائمة. على سبيل المثال، تستطيع ال Child window إضافة علامة صح لأمر ال Options!Save بعبارة مثل هذه:

MainForm.OptionsSave.Checked := True;

والخيلة هنا هي التمكن من الوصول إلى ال Child module. افعل هذا بإضافة Main (أو أى اسم unit أخرى) لعبارة ال Uses فى ال Child module. لمنع ال reference ال cride، يجب أن تفعل هذا فى تنفيذ ال Child window، وليس فى عبارة ال Uses فى الجزء ال interface. ولأن عبارة ال Uses لل Main module تشير بالفعل لل Child module، فإن الإشارة مرة أخرى لل Main فى ال Child window تؤدي إلى ال reference المعادة، والذي تسمح به ال Object Pascal. فى ال Child module، قم بوضع كلمة التنفيذ الرئيسية وأمر تحميل ال form، وأضف عبارة Uses كما هو موضح هنا. يمكنك عندئذ الوصول إلى عناصر قائمة فى ال MainForm:

```
implementation
{$R *.DFM}
uses Main;
```

ونحدث ال unit references ال Cricle عندما تقوم أثنان من modules - ولنسميهما Chicken وEgg - بالإشارة إلى بعضهما فى عبارات ال Uses لقطاعات واجهة التطبيق الخاصة بهما. وتسمح ال Object Pascal بهذا بسبب امكانية أن تستخدم ال Chicken module كتعريف فى ال Egg module، والذي قد يتطلب تعريفاً من ال Chicken module. لحل تناقض ال Chicken-and-Egg، اضع تركيبة ثانوية إلى عبارة Uses فى القطاع interface لل module الأساسية، اضع التركيبة الأساسية إلى عبارة Uses فى قطاع ال implementation للتركيبة الثانوية. ويمكن ألا تشير Unit interfaces إلى بعضها البعض (circularly)، ولكن القطاعات ال implementation الخاصة بها قد تفعل هذا وبلا حدود.

## تقنيات MDI أخرى:

أثناء كتابة هذا الباب، مررت بكثير من أجزاء ال MDI ضمن نماذج ال source code التي شرحتها. فيما يلي بعض تقنيات ال MDI الأخرى التي قد تجدها نافعة.

### الوصول إلى نوافذ الإطار وال client:

قد يكون ضرورياً في بعض الأوقات أن تتمكن من الوصول إلى نوافذ ال client والإطار لتطبيقات ال MDI. إنك لن تحتاج غالباً إلى استخدام هذه التقنيات إلا إذا استدعيت ال Windows API التي تتطلب window handles. يجب أن توفر ال TForm class الخاصة بـ Delphi خدمات كافية لأغلب تطبيقات ال MDI.

استخدم خاصية ال ClientHandle في ال form الرئيسية للإشارة إلى نافذة ال client. في البرمجة الكلاسيكية لل MDI وتوفر هذه النافذة عمليات عامة - مثل التحكم في الوسائل - التي تنطبق على كل ال Child window.

**ملحوظة:** إن قيمة ال ClientHandle، من نوع ال (window) HWND (handle)، تكون صالحة قط في forms ذات ال FormStyle محددة بـ fsMDIForm.



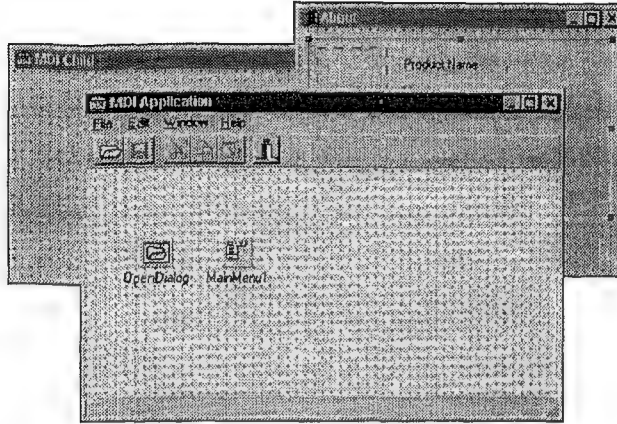
يمكنك الإشارة إلى نافذة الإطار باستخدام خاصية ال Handle لل form الرئيسية. وهذه القيمة دائماً تذكر Window Handel لل form في كلا من تطبيقات ال MDI وغيرها. وفي form ذات ال FormStyle مساوية لـ fsMDIForm، تشير خاصية ال Handle إلى نافذة إطار ال MDI.

### استخدام ال MDI Application Template:

للبرمجة السريعة لأنماط ال MDI النموذجية، استخدم ال MDI Application Template الخاصة بـ Delphi. أولاً، قم بإنشاء دليلاً خالياً لتخزين ملفات المشروع. بعد ذلك، اختر File|New...، اضغط ال Projects page tab في نافذة ال New Items dialog، واختر ال MDI Application. اضغط OK ثم في dialog إختيار الدليل الجديد حيث تريد تخزين ملفات المشروع.



أغلق نافذة dialog الدليل ، ويقوم Delphi بإنشاء تطبيق MDI له status toolbar ، panel وقائمة ذات بنود File ، Edit ، Window ، Help . يوضح شكل ١٥-٣ نافذة ال form الناتجة إلى أعلى نوافذ ال Child وال About اللذان ينشئهما ال template أيضاً .



شكل ١٥-٣: تنشئ ال form MDI Application Template لها status panel ، toolbar وقائمة رئيسية. وتنشئ أيضاً نوافذ ال Child و About والظاهرة هنا مختفية جزئياً خلف ال form الرئيسية.

الإرشادات التالية تساعدك على البدء في استخدام ال MDI Application Template :

\* ال ChildWin.pas module تعرف ال TMDIChild يمكنك إضافة ال LoadData procedure وال SaveData في هذه ال class ، أو يمكنك قراءة وكتابة بيانات وال Main module . إذا كان تطبيقه له نوع Child window واحدة ، يمكنك إضافة البرمجة إلى ال Main ، وإن لم يكن ، إجعل كل class نافذة مسئولة عن مدخلاتها ومخرجاتها ، وأسس calsses نافذة أخرى على ال TMDIChild .

\* حسب النظام الافتراضي للبرنامج ، يتم تسمية ال Application بـ MDIApp . لتعيد تسميته ، اختر File|Save Project ، وإدخل اسم ملف جديد . قم بحفظ وإجراء عملية ال compile للمشروع . يمكنك عندئذ حذف كل ملفات ال MDIApp.\* من دليل المشروع .

## الباب الخامس عشر : تطوير تطبيقات الـ MDI

\* الـ Main.pas module تعرف الـ TMainForm class . ملحوظة :  
راجع تعريفات الـ class في الـ module للحصول على أسماء الـ object متنوعة .  
بعض الأسماء ، مثل Edit1 ، تكون افتراضية طبقاً للبرنامج ، والبعض الآخر ، مثل  
SaveBtn ، ويتم تسميتها طبقاً للإستخدام . إن تقاليد التسمية الخاصة بالـ  
template ليست كما تسمى التطبيقات في هذا الكتاب . يمكنك تغييرها ، ولكن إذا  
فعلت هذا ، يجب أن تغييرها في كل مكان تحدث به . فليس من الكافي ، مثلاً ، أن  
تغير خاصية الـ Name كـ object - فيجب عليك أيضاً أن تبحث وتستبدل بصفه  
عامة اسم الـ object في أى عبارات تشير إليه . يقوم Delphi بصورة تلقائية  
بتحديث تعريفات اسم الـ object فقط ، فهو لا يبحث عنه ويستبدل أسماء الـ  
object في العبارات .

\* اختر الـ ViewProject Manager وقم بإبراز الـ ChildWin لفتح الـ  
form و Child window unit . يمكنك عندئذ إدخال أية objects تريدها في الـ  
Child window ، إنشاء قائمة مدمجة ، وكتابة code لأوامر الـ Child  
window .

\* في الـ Main module ، أضف عبارات الـ data-loading  
للـ FileOpenItemClick procedure . كما ذكرت ، يمكنك قراءة ملفات  
مباشرة ، أو يمكنك إستدعاء الـ LoadData procedure لنافذة  
الـ ActiveMDIChild .

\* أضف الـ SaveDialog على الـ form الرئيسية ، وأكتب code لحفظ بيانات  
في الـ FileSaveItemClick . أو ، يمكنك إستدعاء الـ SaveData procedure  
الذي أضفته لـ class النافذة الصغيرة .

\* إختيارياً ، قم ببرمجة أوامر الـ Cut ، Copy ، و Paste بإستدعاء  
الـ clipboard procedures . على سبيل المثال ، يمكنك إدخال Memo في الـ  
ChildWin form ، وإستدع الـ CutToClipboard method للـ object . أنظر  
الباب السادس عشر لمزيد من المعلومات عن برمجة الـ clipboard .

\* أدخل الـ OnCloseQuery في الـ ChildWin module . حدد الـ  
CanClose بـ False لمنع المستخدمين من إغلاق نوافذ دون حفظ التغييرات .

## افكار للمستخدم الخبير

\* نافذة واحدة فقط في تطبيق ال MDI يمكن أن تكون ال FormStyle الخاصة بها محددة بـ fsMDIForm . ويجب أن تكون هذه النافذة الرئيسية للتطبيق (إستخدم ال ProjectOptions إذا لزم الأمر لتحديد أي form هي الرئيسية) .

\* إذا ظهرت ال Child window الخاصة بالبرنامج كنوافذ مستقلة على شاشة الحاسب المكتبي ، بدلاً من أن تظهر داخل النافذة الرئيسية ، فإن خاصية ال FormStyle لـ Child window تكون غالباً غير محددة بشكل سليم . (في النسخ السابقة من Delphi و Windows) يمكن أن تقود هذه المشكلة إلى خطأ حماية عامة ، ولكن الأمر لم يعد كذلك) . يجب أن تكون كل ال Child forms window لها خاصية ال FormStyle محددة بـ fsMDIChild . إن النافذة الرئيسية للبرنامج يجب أن يكون ال FormStyle الخاصة بها محددة بـ fsMDIForm ، ويجب أن تكون نافذة واحدة لكل تطبيق تملك هذه الصفة .

\* إذا لم تكن تريد أن تظهر ال Child window بصورة تلقائية عندما تقوم بتشغيل التطبيق ، اختر OptionsProject وأنقل ال Child window إلى قائمة ال Available forms . ولكن ، لا يوجد خطأ من الناحية الفنية في أن يقوم التطبيق بإنشاء ال Child window الأولى تلقائياً . على سبيل المثال ، يمكن أن يمنع محرر نص ال MDI المستخدمين نافذة خالية في البداية .

يستطيع التطبيق إنشاء ال Child window إضافية كما هو موضح في هذا الباب - يتم إنشاء النافذة الأولى فقط بصورة تلقائية عندما يبدأ البرنامج .

\* إذا كان إغلاق ال Child window لتطبيق ال MDI يجعلها تصغر ، تأكد من أن module ال Child window لها ال OnClose الذي يحدد ال Action بـ caFree . إذا لم تقم بتعيين هذه القيمة للـ Action ، لا يقوم التطبيق بتحرير ال object أخرى في وقت التشغيل (مثل ال TChildForm في هذا الباب) ، تذكر تعيين حالات ال object التي تم إنشاؤها في المتغير الحالي - لا تستدع ال Create method بإستخدام المتغير الحالي الذي لم يتم بدؤه . على سبيل المثال ، هذا الخطأ كثيراً ما يحدث :

```
var
  Child: TChildForm;
begin
  Child.Create(Application); {???}
...

```

والعبارة الخاطئة تحاول أن تجعل ال Child window تولد نفسها ، والذي لا يمكن أن يحدث . وغالباً ما تؤدي العبارة إلى خطأ "access violation" أو إنتهاك الوصول لأن ال Child لم يبدأ واستدعاء object methods لم يتم بدئه ، يعد ذلك خطأ برمجة . إستخدم ال code التالية لتستدعي ال Create method لل class وتعين ال object الناتج إلى ال Child reference :

```
var
  Child: TChildForm; // Or, you may use TForm as the type
begin
  Child := TChildForm.Create(Application);

```

\* ليس مطلوب منك أن تحفظ object references نافذة صغيرة مع ال Application كنافذة أم يؤدي إلى إنشاء النافذة الصغيرة . إن النافذة الأم مسئولة عن المحافظة على وتحرير ال object الناتج . وكمثال على هذا المبدأ ، يمكنك تقصير ال CreateChild procedure في ال Main module لل MDIDemo للآتي . ولكن ، هذا يجعل من الصعب أن تؤدي عمليات على ال Child window الحديثة مثل تعيين Caption strings :

```
procedure TMainForm.CreateChild(const Name: String);
begin
  TChildForm.Create(Application);
end;
```

\* في تطبيقات ال MDI ذات الأنواع المتعددة من ال Child window ، قم بتسمية ال unit modules ب ChildXXX.pas ، حيث أن ال XXX هي إمتداد اسم ملف الوثيقة . على سبيل المثال ، إستخدم ال Childtxt.pas للملفات ال txt . أو Childbmp.pas للملفات ال bmp .

بالطبع يمكنك أن تسمى ال modules كما تشاء ولكن هذا تقليد مفيد لتعريف ال modules بأنواع ملفاتها .

\* يكون دمج القوائم فى تطبيقات ال MDI تلقائياً . حدد كل خصائص MainMenu AutoMerge بـ False . يجب أن تكون خاصية ال AutoMerge لنافذة ال MDI الأم محددة بـ False . إستخدم خاصية ال AutoMerge للقيام بدمج القوائم فقط فى غير تطبيقات ال MDI .

\* تجنب إسقاط components على ال form الرئيسية لل MDI . إن ال controls خفيفة الوزن مثل ال TLabel لن ترسم من الأساس ، و handle controls النافذة العادية (مثل ال TButton) قد لا تعمل بشكل صحيح عند إسقاطها على client area لل MDI form الرئيسية . وهذا بسبب التلاعب فى واجهة التطبيق لل MDI Windows . ولكن ، ال TPanel العلوية أو السفلية المعنية تعتبر مقبولة . أضف control objects أخرى على ال Child window أو فى ال Panel objects .

### المستروعات التى يمكنك تجربتها

- ١-١٥ : أضف Child window metafile لل MDIDemo .
- ٢-١٥ : أكتب تطبيق تصفح ملف يعمل مع مجموعة متنوعة من الملفات .
- ٣-١٥ : قم بتصميم أمر FileSave All ظاهرى فى Child class window .
- ٤-١٥ : متقدم . أضف أمر Window|Restore لتطبيق ال MDIDemo . يجب أن يقوم الأمر بإستعادة Child window إلى مواضعها وأحجامها الأولى عقب أمر Tile أو Cascade أو Minimize . all تذكر إستخدام البرمجة المختصة بال object يجب أن تكون Child window objects قادرة على إستعادة نفسها إلى موضع وحجم محفوظ . لا تكلف الأم بهذا العبء؛ لجعل الصغار يعملون .

## الباب الخامس عشر : تطوير تطبيقات الـ MDI

٥-١٥ : أكتب محرر نص MDI باستخدام MDI Application

template . لن تكون النتائج خارقة ، ولكن التجربة ستجعلك تفهم تقاليد التسمية المتنوعة للـ template .

٦-١٥ : متقدم . قم بتحويل الـ MDIDemo إلى MDI Application

template محسن إدخال الـ virtual methods مثل الـ LoadData والـ SaveData في الـ TChildForm class لا تقم بإنشاء object من تلك الـ class ، بدلاً من ذلك ، قم بتقسيم فرعي لـ forms جديدة على أساس الـ TChildForm وأكتب الـ LoadData و SaveData procedures لقراءة وكتابة أنواع معينة من الملفات . لكي تبرمج تطبيقاً جديداً ، فإنك تضيف form Child window جديدة ، أكتب LoadData proceducers و SaveData و قم بإنشاء نماذج صغيرة في الـ Main module .

### ملخص :

\* إن تطبيقات الـ MDI تستخدم الـ TForm الخاص بـ Delphi ، وهو غير موجود على لوحة الـ VCL palette ، لإنشاء نوافذ رئيسية و Child window . وتحل النافذة الرئيسية محل نوافذ الـ client والإطار المستخدمة في برمجة الـ Windows الكلاسيكية .

\* إن كل تطبيق MDI له ثلاثة أجزاء أساسية : form نافذة رئيسية ، form واحدة أو أكثر من Child window الـ Client ، وقائمة رئيسية .

إستخدم خاصية الـ WindowMenu للـ form الرئيسية لتحديد بند قائمة رفيعة المستوى لعناوين نافذة القائمة .

\* حدد خاصية الـ FormStyle للـ form الرئيسية بـ fsMDIForm . يمكن أن يكون هناك form واحدة فقط من هذا النوع في كل تطبيق . في كل Child window ، حدد الـ FormStyle بـ fsMDIChild يمكنك إنشاء أي عدد من أنواع Child window كما يحتاج تطبيقك .

\* لإنشاء Child window فى وقت التشغيل ، قم بإستدعاء ال Create method الخاص بـ class النافذة قم بتمرير ال Application إلى ال object parameter الأم للـ Create method .

\* يجب أن تقوم كل Child window بتنفيذ ال OnClose و OnCloseQuery إستخدم هذه ال events لتحذر المستخدم حول البيانات التى لم يتم حفظها عندما يحاول إغلاق النوافذ .

\* إن خاصية ال ActiveMDIChild الخاصة بالـ form الرئيسية تشير إلى Child window الحالية . إذا كانت هذه الخاصية nil ، فلا يوجد Child window . قم دائماً بفحص ما إذا كانت ال ActiveMDIChild مساوية لـ nil قبل إستخدام هذه الخاصية لإستعداد method وتعيين قيم خاصية .

\* إن خاصية ال MDIChildCount التابعة للـ form الرئيسية تساوى عدد النوافذ الصغيرة . إذا كانت ال MDIChildCount مساوية لصفر ، فلا يوجد Child window .

\* إن ال MDIChildren array يوفر وصولاً مفهراً لكل ال Child window المملوكة للتطبيق .

\* إن ال MainMenu للـ Child window تندمج بصورة تلقائية مع ال MainMenu الخاص بالـ form الرئيسية . إستخدم خاصية ال GroupIndex للتحكم فى الدمج .

\* يشير ال Handle الخاص بالـ form الرئيسية إلى نافذة الإطار لتطبيق ال MDI . تشير خاصية ال ClientHandle إلى نافذة العميل . لا تحتاج أغلب تطبيقات ال MDI إلى إستخدام هذه الخصائص .

\* إستخدم template مشروع ال MDI Application حتى تنشئ بسرعة مشروعات ال MDI ذات قوائم افتراضية ، toolbar ، و statusbar .

إن نقل البيانات من تطبيق إلى آخر يعتبر أمراً صعباً فى برمجة Windows . إن الباب التالى يساعدك على خوض ما لم تقنيات نقل بيانات ال clipboard ، وال DDE ، وال OLE . وهو أيضاً يوضح إستخدام وإنشاء ال controls ال ActiveX .

\*\*\*\*\*

# الباب السادس عشر

## التطوير مع الـ Clipboard، الـ DDE، والـ OLE

### محتويات هذا الباب:

- Components.
- نقل بيانات الـ Clipboard.
- الـ TClipboard class.
- Dynamic Data Exchange (DDE).
- Object linking and embedding (OLE).

إن تشارك البيانات بين التطبيقات أمراً ليس سهلاً. يمكنك أن تقود code إلى بيئة شبكية أو متعددة المهام، ولكن أن تجعلها تتشارك في البيانات مع برامج أخرى، فهذا يأخذ مزيداً من العمل والمراوغة.

يوفر الـ Windows ثلاثة methods أساسية للتشارك في البيانات: clipboard، الـ Dynamic Data Exchange (DDE)، والـ Object Linking and Embedding (OLE). وأي method تستخدمها فهذا يعتمد على احتياجات تطبيقك ونوع البيانات التي يجب عليك أن تتشارك فيها. في هذا الباب، سوف أوضح كيفية استخدام الـ clipboard لنقل النص والجرافيك، وكيفية استخدام الـ DDE لإقامة روابط مع برامج مثل الـ Microsoft Excel و Word، وكيفية ربط وتضمين الـ OLE في تطبيقات Delphi.



## : Components

فيما يلي قائمة بـ Delphi Components للتطوير مع clipboard والـ DDE والـ OLE :

• Dde ClientCouv - إن تطبيق عميل DDE (وهو تطبيق يتلقى بيانات من مساعد الـ DDE) يستخدم هذا الـ Component لإقامة رابطة، تسمى محادثة [أو conversation]، مع server. كل تطبيق client له على الأقل واحداً من هذه الـ PaletteSystem ibject .

• Dde Client Item : بالإضافة إلى الـ Dde Client Couv object، يستخدم تطبيق الـ DDE client واحداً أو أكثر من الـ Dde Client Item كحاويات تتلقى البيانات من الـ server. كل تطبيق client له على الأقل واحداً من هذه الـ objects لكل Palette: System . DdeClientConv

• DdeServerConv - إن تطبيق (server) DDE (تطبيق يرسل بيانات لـ DDE client) يمكن أن يستخدم هذا الـ component لخلق محادثة DDE. إن أغلب تطبيقات الـ server تستخدم على الأقل واحداً من هذه الـ objects، ولكن من الممكن خلق محادثة بدون الـ Palette: System . DdeServerConv.

• DdeServerItem - يستخدم تطبيق أو الـ Server الـ DDE واحداً أو أكثر من هذه الـ objects لإرسال بيانات للـ client. إن الـ DdeServerItem يمكن ربطه أو عدم ربطه بالـ DdeServerConv، ولكنه غالباً ما يكون مرتبطاً به. Palette: System .

• OleContainer - استخدم هذا الـ object لإنشاء حاوية OLE يمكن أن تحمل objects من الـ OLE servers. على سبيل المثال، باستخدام الـ OleContainer، يمكن لوثائق تطبيقك أن تجمع لوحات الـ Excel، ووثائق الـ Word، ورسوم الـ Visio Express. يستطيع المستخدمون عرض بيانات في هذه الـ objects مباشرة من تطبيقك. Palette: System .

## انتقالات البيانات عبر الـ Clipboard:

كما يعرف أغلب مستخدمي الـ Windows، تقوم أوامر الـ Copy، Cut، Paste الخاصة بقائمة الـ Edit بنقل البيانات من وإلى الـ Windows clipboard.

## الباب السادس عشر : التطوير مع ال Clipboard، ال DDE، وال OLE

وغالباً ما يكون الاستخدام الشائع لهذه الأوامر هو نقل النص بين التطبيقات، أو حتى داخل نفس التطبيق، ولكن Clipboard يمكن أن تتولى أمر نقل البيانات في Forms أخرى لا حصر لها. على سبيل المثال، يستطيع المستخدمون قص ولصق ال bitmaps بين برامج الجرافيك.

إن ال Edit وال MaskEdit وال Memo التابعة لـ Delphi (وكذلك ال TDBEdit components، و TDBImage، و TDBMemo) تدعم نقل البيانات من خلال ال clipboard لنص احادي العبارة. لنقل بيانات ال object، استدع methods مثل ال CopyToClipboard، CutToClipboard، PasteToClipboard، إستجابة لإختيار المستخدم لأمر قائمة أو ضغط زر. على سبيل المثال، لقص نص مختار من ال Memo، استخدم العبارة:

```
Memo1.CutToClipboard;
```

وال two procedures الأخران أيضاً يسهل استخدامهما:

```
Memo1.CopyToClipboard;
```

```
Memo1.PasteFromClipboard;
```

**فكرة:** ال components المؤسسة على نص، فإن النص المختار فقط يتم نسخة أو قصة إلى ال clipboard. لنسخ أو قص كل text object، استدع Memo1.SelectAll قبل أداء عملية ال clipboard.

### ال TClipboard Class:

لقص ولصق بيانات غير نصية، استخدم ال TClipboard Class في ال Clipbrd unit (لاحظ الهجاء المختصر). لأن ال TClipboard ليست component على ال VCL palette، فلكى تستخدم هذا ال class يجب ان تضيف اسم ال unit لتعريف ال uses لل module. قم بتعديل التعريف كما يلي:

Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, Clipbrd;

### خصائص ال TClipboard:

توفر ال TClipboard الخصائص التالية:

• **AsText** - استخدم خاصية الـ string هذه لنسخ ولصق النص من وإلى الـ clipboard ، قم بتعيين Pascal string للـ AsText . ولكي تتلقى الـ string ، استخدم الـ string فى عبارة . لقراءة وكتابة نص clipboard اطول ، يمكنك أيضاً ان تستخدم الـ SetTextBuf methods والـ GetTextBuf للـ TClipboard ولكن لأن الـ Object Pascal strings يمكن أن تكون الآن غير محددة فى الطول ، فهذه الـ methods لم تعد على نفس درجة الإفادة كما كانت فيما مضى . إذا كانت الـ clipboard تحتوى على نص - فمن الحكمة دائماً أن تفحص هذا قبل اللصق - فتكون عبارة الـ If التالية صادقة :

if Clipboard.HasFormat(cf\_Text) then ...

• **FormatCount** - إن خاصية العدد الصحيح هذه تساوى عدد العناصر فى الـ Formats array ، الموضح بعد ذلك . إن تعبيرات الـ array الصالحة تتراوح فيما بين الـ Formats[0] إلى [FormatCount - 1] . استخدم دائماً الـ FormatCount لمنع اخطاء مدى الفهرس عند استخدام الـ Formats array .

• **Formats** - هذا هو Array من قيم الـ Word التى تمثل Forms أو أنواع البيانات المسجلة الخاصة بالـ clipboard . إن خاصية الـ FormatCount تساوى عدد العناصر فى الـ Formats array . يذكر الجدول ١٦-١ ثوابت الـ clipboard format المعيارية من الـ Windows.pas المتوفر فى الـ source files لـ Delphi . توجد الثوابت فى الحروف الكبيرة بالملف ، ولكن يمكنك أن تكتبها بالحروف الصغيرة إذا أردت .

### تنسيقات إضافية:

تسجل الـ Clipbrd unit اثنين من تنسيقات الـ clipboard بالإضافة إلى التنسيقات المعيارية المذكورة فى جدول ١٦-١ . تقوم الـ unit بهذا عن طريق تنفيذ هذه العبارات فى أى برنامج يستخدم الـ Clipbrd :

```
cf_Picture := RegisterClipboardFormat('Delphi Picture');
cf_Component := RegisterClipboardFormat('Delphi
Component');
```

<i>Constant</i>	<i>Format Value</i>
CF_TEXT	1
CF_BITMAP	2
CF_METAFILEPICT	3
CF_SYLK	4
CF_DIF	5
CF_TIFF	6
CF_OEMTEXT	7
CF_DIB	8
CF_PALETTE	9
CF_PENDATA	10
CF_RIFF	11
CF_WAVE	12
CF_UNICODETEXT	13
CF_ENHMETAFILE	14
CF_HDROP	15

استخدم متغيرات ال cf\_Picture وال cf\_Component، مع ال HasFormat الخاصة بال clipboard، لتحديد إذا ما كانت ال clipboard تحتوي الآن على صورة TGraphics (أيقونة، bitmap، أو ملف حجم كبير) أو نوع آخر من ال component object. إن Delphi نفسه يستخدم cf\_Component لقص ولصق ال component objects على ال forms، ولكن التطبيقات لا تحتاج إلى استخدام هذا المتغير.

وهذا هو مثال لـ custom clipboard format، والذي يمكنك ان تفعله في تطبيقاتك. على سبيل المثال، استخدم هذه التقنيات لتسجيل structure format

معين لنقل بيانات من هذا النوع إلى تطبيقاتك أخرى باستخدام ال clipboard object .

### ال TClipboard methods :

استدع ال TClipboard class لقص ونسخ ولصق بيانات، وكذلك لأداء عمليات clipboard متنوعة. توفر ال Clipbrd unit ال Clipboard object جاهز للاستخدام- لا يجب عليك إنشاء object من ال TClipboard class. أضف لأمر ال Uses ال module. يمكنك ان تفعل هذا بواسطة تعديل تعريف ال Uses لل module. على سبيل المثال، اضع النص التالي (الموضح بالخط السميك):

implementation

uses Clipbrd

**تحذير:** انتبه إلى كيفية هجاء ال Clipbrd. لا تفعل كما فعلت أنا - كتبت **uses Clipboard**، ثم أضعت عشر دقائق اتعجب لما لم يتم برنامجي بال compile.

Caution

إن استخدام ال Clipbrd unit بهذه الطريقة يجعل أى تعريف عام متاح لل procedures وال functions الموجهة فى قطاع ال implementation module. على سبيل المثال، فى procedure أو function، يمكن لعبارة أن تستدعى ال HasFormat method الخاص بال Clipbrd unit مثل هذه:

if Clipboard.HasFormat(cf\_Bitmap) then

// ... copy bitmap from clipboard

لاحظ ان الاستدعاء لل HasFormat هو فى اشارة لل Clipboard، والذي قد تم بدئه بالفعل وجاهز للاستخدام. إذا لم تكن تريد أن تكتب Clipboard مرات ومرات، يمكنك استخدام عبارة with كما فى هذه الجزئية:

with Clipboard do

begin

if HasFormat(cf\_Bitmap) then

// ... copy bitmap from clipboard

else if HasFormat(cf\_Text) then

```
// ... copy text from clipboard
```

```
end;
```

فيما يلي وصف مختصر للـ TClipboard methods الرئيسية التي يمكنك استدعاؤها بالإشارة للـ Clipboard :

● Assign-To : لنسخ الجرافيك إلى Clipboard ، قم بالتمرير للـ Assign أي TGraphic ، أو TBitmap ، أو TPicture ، أو TMetafile أو أي خاصية object .

● Clear : يسمح لأي بيانات في الـ clipboard . لا يجب عليك عادة ان تستدعي هذا الـ method ، لأن تعيين معلومات جديدة إلى الـ clipboard . يقوم بأداء مسح Clear . ولكن ، قد تستدعي الـ Clear قبل إنتهاء البرنامج مباشرة إذا كانت الـ clipboard بها مجموعة كبيرة من البيانات . إسئل المستخدم عما إذا كانت الـ clipboard يجب مسحها .

● Close : يغلق الـ clipboard عقب استدعاء الـ Open . كل عبارة Open يجب أن يكون لها عبارة Close مقابلة لها . لا يجب عليك استدعاء الـ Open والـ Close إلا إذا وصلت للـ clipboard باستدعاء Windows API . إن الـ Windows API في الـ Clipboard تفتح وتغلق الـ Clipboard بصورة تلقائية على حسب الحاجة .

● GetAsHandle : يُعاد الـ handle إلى بيانات الـ clipboard . استخدم هذا فقط إذا احتجت إلى استدعاء الـ Windows API التي تتطلب handles ، أو عند تحويل البرامج التي كتبت بطريقة تقليدية إلى Delphi . معظم التطبيقات لا تحتاج إلى استخدام هذه الـ function .

● GetComponent : تستعيد الـ Delphi component من clipboard . لمزيد من المعلومات حول استخدام الـ GetComponent والـ SetComponent انظر الـ Component objects والـ clipboard ، في هذا الباب .

● GetTextBuf : هذه الـ function تستعيد النص من الـ clipboard كـ null-terminated buffer . في الماضي ، كانت هذه الـ method تستخدم مع

الكميات الكبيرة من النص ، لأن الـ Pascal strings ليس لها حد في الطول ، فيمكنك ببساطة ان تستخدم خاصية الـ AsText لاستعادة نص الـ clipboard . " انظر النص و الـ clipboard " في هذا الباب . لمزيد من المعلومات حول استخدام الـ GetTextBuf و الـ SetTextBuf .

• HasFormat : ترجع بـ True إذا كانت الـ clipboard لها بيانات بـ format عُرفت من خلال Word argument دائماً أستخدم HasFormat قبل نسخ معلومات من الـ clipboard في تطبيق أو متغير آخر .

• Open : انك تحتاج لفتح الـ clipboard باستدعاء هذا الـ Windows فقط إذا استدعيت الـ Windows API للوصول الى الـ clipboard . إن كل الـ methods في الـ clipboard تفتح clipboard بصورة تلقائية على حسب الحاجة .  
تحذير : إن استدعاء الـ Open يمنع بشكل مؤقت المهام الأخرى من استخدام الـ clipboard . وكل استدعاء للـ Open يجب أن يكون له استدعاء مقابل للـ Close .

• SetAsHandle : يُعين بيانات للـ clipboard باستخدام Windows handle . انظر الـ GetAsHandle . ان غالبية التطبيقات لا تحتاج إلى استدعاء هذا الـ method .

• SetComponent : يعين component إلى الـ clipboard . انظر الـ GetComponent و Component clipboard في هذا الباب لمعرفة المزيد من المعلومات .

• SetTextBuf : يعين بيانات null-terminated للـ clipboard . استخدم هذه الـ function للعمل مع كميات كبيرة من النص المخزن في الـ Char arrays . انظر أيضاً الـ GetTextBuf . لأن الـ Pascal strings أصبحت الآن فعلياً غير محددة في الطول ، فإن اغلب التطبيقات يمكنها ان تعين string لخاصية الـ AsText التابعة للـ Clipboard بدلاً من استدعاء الـ SetTextBuf و الـ GetTextBuf .

### النص و الـ clipboard :

من السهل نقل النص من والى الـ clipboard والـ components على نص مثل الـ Memo والـ Edit باستخدام نوع بيانات الـ string الديناميكي للـ Pascal ،

فبإمكانك ان تعين أى خاصية Text لخاصية ال AsText التابعة لل Clipboard . ولكن هناك بعض النقاط التى تستحق مزيداً من الاعتبار .

لعرض هذه الجوانب وال clipboard فان الخطوات التالية تستخدم ال Memo . بالرغم من أنه يمكنك بسهولة أكثر ان تؤدى عمليات مشابهة باستدعاء ال CopyToClipboard ، CutToClipboard ، PasteFromClipboard التابعة لل Memo ، فالخطوات التالية تقدم تقنيات برمجة هامة يمكنك تطبيقها على أنواع أخرى من عمليات نقل بيانات ال clipboard . اتبع هذه الخطوات :

١- ابدأ تطبيقاً جديداً . اصف تعريف ال uses لل Clipbrd بعد كلمة ال implementation الأساسية فى ال form unit. للتطبيق . يجب ان يبدأ التطبيق هكذا :

```
implementation
uses Clipbrd;
```

٢- أضف Memo من ال Standard palette على ال form . اختر Memo1 وحدد خاصية ال ScrollBars له ب ssBoth . يمكنك تكبير ال Memo1 إذا أردت .

٣- أضف اثنين من ال Button على ال form . قم بتغيير خاصية ال Name للزر الأول ليصبح CopyButton وال Caption الخاص به ليصبح Copy . قم بتغيير خاصية ال Name للزر الثانى ليكون PasteButton وال Caption الخاص به ليكون Paste . شكل (١٦-١) يوضح نافذة البرنامج حتى الآن .

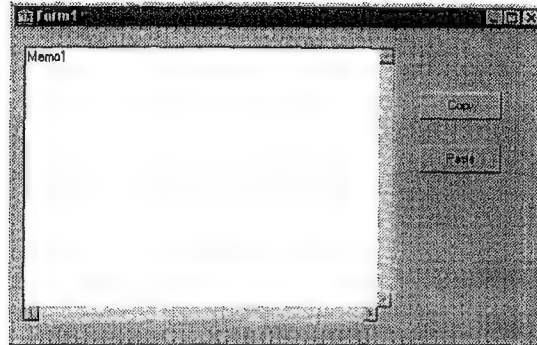
٤- اضغط مرتين زر ال Copy لإنشاء ال onClick الخاص به . أدخل ال code من القائمة (١٦-١) فى ال procedure على القرص المدمج ، يوجد هذا النص فى ملف ال Clip1.pas فى دليل ال Source\Clipboard .

٥- اضغط مرتين زر ال Paste لإنشاء ال onClick الخاص به . أدخل ال code من القائمة (١٦-٢) فى ال procedure على القرص المدمج ، يوجد هذا النص فى ملف ال Clip1.pas فى دليل ال Source\Clipboard .

٦- قم بتشغيل البرنامج الاختبارى واستخدم ال Windows Notepad utility لفتح ملف نص . اختر كل السطور ، اختر ال Edit|Copy لنقل النص إلى



clipboard. ارجع مرة أخرى إلى البرنامج الاختباري، واضغط زر الـ Paste لنسخ نص الـ clipboard إلى الـ Memo. اضغط Copy لنسخ النص إلى الـ clipboard. احذف النص، واضغط Paste للتحقق من أن البرنامج يلصق الـ clipboard إلى الـ Memo. يمكنك أيضاً أن ترجع للـ Notepad، وتختار الـ Edit|Paste لإدخال النص المنسوخ من الـ clipboard.



شكل (١٦-١) الـ Form window الخاصة باختبار الـ clipboard مع الـ Memo والـ Buttons.

القائمة ١٦-١، الـ Click للزر الـ Copy.

```
procedure TForm1.CopyButtonClick(Sender: TObject);
begin
  with Memo1 do
    begin
      if SelLength = 0 then // If no text selected,
        SelectAll;         // select all lines of Memo1.
        if SelLength = 0 then // If still no text selected,
          Exit;             // Memo1 is empty-exit now.
        Clipboard.Clear;
        Clipboard.AsText := Memo1.Text;
      end;
    end;
end;
```

القائمة ١٦-٢:OnClick لزر ال Paste.

```
procedure TForm1.PasteButtonClick(Sender: TObject);
begin
  if Clipboard.HasFormat(CF_TEXT) then
    Memo1.Text := Clipboard.AsText;
end;
```

تقنية نص clipboard،

هناك تقنية قديمة للعمل بنص clipboard والتي لا تزال متاحة . يمكنك استخدام هذا ال method إذا كنت تكتب code للـ Windows 3.1 ، أو إذا كان لديك نسخة أولية من Delphi . يمكنك أيضاً استخدام المعلومات الواردة في هذا الفصل لتحديث البرامج القديمة التي تستخدم هذه التقنية حالياً لتخزين نص ال clipboard في ال character arrays .

استدع ال method SetTextBuf الخاص بال Clipboard لتنسخ Buffer النص الكبير إلى ال Clipboard . على سبيل المثال ، إذا كان ال PChar Buffer يشير إلى ال null-terminated string buffer ، فهذه العبارة تنسخ النص إلى ال clipboard .

```
Clipboard.SetTextBuf(Buffer);
```

لاستدعاء النص من ال clipboard ، قيم بتخصيص ذاكرة للـ buffer واستدع ال GetTextBuf function الخاصة بال Clipboard ، والتي ترجع عدد ال Clipboard المنسوخة .

```
Buffer := StrAlloc(1024);
```

```
Len := Clipboard.GetTextBuf(Buffer, 1024);
```

**على القرص المدمج:** لسوء الحظ ، إن ال GetTextBuf يطلب منك ان تحدد مسبقاً حجم ال buffer النص ، ولكن ال Clipbrd unit لا تقدم أية methods لتحديد هذا الحجم . لمعرفة السبيل إلى الخروج من هذه المشكلة ، اتبع الخطوات المذكورة في الفصل السابق لإنشاء برنامج اختباري له Memo واثنين من ال Buttons يسميان CopyButton و PasteButton . بدلاً من



استخدام OnClick للـ Button فى القائمة السابقة ، استخدم الـ code الموجودة فى القائمة ١٦-٣ والقائمة ١٦-٤ . يوجد هذا النص على القرص المدمج فى دليل .Source\Clipboard

**القائمة ١٦-٣: OnClick هذا يعرض كيفية استخدام الـ SetTextBuf method الخاص بالـ Clipboard لنسخ null-terminated string buffer إلى الـ clipboard.**

```

{ Copy null-terminated text buffer to clipboard }
procedure TForm1.CopyButtonClick(Sender: TObject);
var
  P: PChar; { Pointer to character buffer }
begin
  with Memo1 do
    begin
      if SelLength = 0 then // If no text selected,
        SelectAll;        // select all lines of Memo1.
        if SelLength = 0 then // If still no text selected,
          Exit;             // Memo1 is empty-exit now.
          P := StrAlloc(SelLength + 1); // Allocate char
          buffer
          try
            GetTextBuf(P, SelLength + 1); // Copy text to buffer
            Clipboard.SetTextBuf(P); //Copy text buffer to
            clipboard
            finally
              StrDispose(P); // Dispose of character buffer
            end;
          end;
        end;
      end;
    end;
  end;
end;

```

**القائمة ١٦-٤: OnClick هذا يعرض بديل للـ GetTextBuf method بالـ Clipboard، والذي يتطلب منك ان تحدد حجم الـ buffer مسبقاً.**

```

{ Copy null-terminated text buffer from clipboard }
procedure TForm1.PasteButtonClick(Sender: TObject);
var
  Data: THandle;

```

## الباب السادس عشر : التطوير مع ال Clipboard، ال DDE، وال OLE

```
DataPtr: PChar;  
P: PChar;  
begin  
  Clipboard.Open;  
  try  
    Data := GetClipboardData(cf_Text);  
    if Data = 0 then Exit;  
    DataPtr := GlobalLock(Data);  
    try  
      P := StrNew(DataPtr);  
      Memo1.SetTextBuf(P);  
    finally  
      GlobalUnlock(Data);  
      StrDispose(P);  
    end;  
  finally  
    Clipboard.Close;  
  end;  
end;
```

تأكد القائمة ١٦-٣ من أن ال Memo1 له بعض النص المختار- إذا لم يكن، يقوم ال procedure باستدعاء ال SelectAll. هذه الخطوة اختيارية، ولكنها تجعل من السهل ان تنسخ كل الوثيقة إلى ال clipboard دون ان تطلب من المستخدمين ان يختاروا كل رمز. يقوم ال StrAlloc بأعداد ال buffer الذي ينسخ ال GetTextBuf بداخله نص ال Memo1. يستدعي البرنامج بعد ذلك ال SetTextBuf الخاصة بال Clipboard لنسخ ذلك ال buffer إلى ال Windows clipboard.

عند القيام بتخصيص أماكن في الذاكرة، من الحكمة دائماً ان تدخل عبارات متابعة في ال try-finally. كما توضح القائمة ١٦-١، فإن الذاكرة المخصصة للـ P تم تنظيمها بواسطة ال StrDispose بغض النظر عما إذا كانت أي عبارة في ال try تثير exception.

وتدور القائمة ١٦-٤ حول ال Clipboard lack ال method لتحديد كم النص الذي تحمله ال clipboard. فبدلاً من استدعاء ال GetTextBuf لاستدعاء

النص من ال clipboard ، يقوم ال procedure باستدعاء ال Windows API للحصول على هذه ال Windows API ، وهى ال GetClipboardData .

إن ال try الداخلى لل procedure يستدعى ال StrNew لتخصيص ذاكرة لل P ال PChar pointer ، ثم ينسخ بيانات ال clipboard ك null-terminated string إلى تلك الذاكرة . وللعرض فقط ، يقوم ال procedure بتمرير ال string لل Memo .

**تحذير:** عند استخدام نقل ال clipboard method فى القائمة ١٦-٤ ، يجب أن تستدعى ال Open وال Close كما هو موضح لأن ال procedure يمر على ال Clipboard methods . تذكر ، التقنيات المذكورة فى هذا الفصل توضح ال methods - لا تستخدمها فى تطبيقات جديدة .



### الجرافيك وال Clipboard

يمكنك استخدام ال Assign method لل Clipboard لقص ، نسخ ، ولصق ال TGraphics مثل أولئك الموجودون فى خاصية ال Picture لل Image . لتوضيح التقنية ، إبدأ تطبيقاً جديداً أو أضف تعريف Uses إلى قطاع ال implementation  
uses Clipbrd;

اضف اثنين من ال Image من Additional palette و Button من Standard palette على form . حدد خاصية ال Stretch لل Images بـ True . اضغط مرتين ال Button وادخل العبارات التالية بين ال begin وال end فى ال OnClick لل object . (سوف تؤدي هذه العمليات فى اماكن منفصلة فى البرامج أو فى تطبيقات منفصلة . ولكن للعرض ، قم بكلا الانتقالين فى وقت واحد) .

```
Clipboard.Assign(Image1.Picture); // Copy to clipboard
if Clipboard.HasFormat(cf_Picture) then
    Image2.Picture.Assign(Clipboard); // Copy from clipboard
```

مازلنا فى Delphi ، اختر Image1 واستخدم خاصية ال Picture التابعة له لتحميل ايقونة ك Bitmap ، أو ملف metafile . قم بتشغيل البرنامج بضغط F9 ،

## الباب السادس عشر : التطوير مع الـ Clipboard، الـ DDE، والـ OLE

ثم اضغط الزر لنقل الصورة من Image1 الى Image2 عبر Clipboard الـ Windows . يجب ان يعرض كلا من الـ Images نفس الجرافيك .

تستدعي العبارة الأولى الـ Clipboard Assign method لنسخ خاصية TGraphics (Image1.Picture) الى الـ Clipboard . تتحقق العبارة الثانية من ان الـ Clipboard لها cf\_Picture ، وإذا كان كذلك ، تستدعي الـ Picture.Assign method الخاص بالـ Image2 ، مما يمرر الـ argument . بالرغم من انك قد تستخدم الـ Clipboard على انها الـ Assign argument بهذه الطريقة ، فيجب دوماً ان تختبر كما هو موضح هنا وذلك باستدعاء الـ HasFormation ما اذا كانت بيانات الـ Clipboard موجودة في الـ form المتوقعة . على سبيل المثال ، لتلقى Bitmap منسوخة إلى الـ Clipboard بواسطة تطبيق آخر مثل Windows Paint utility ، استخدم Code مثل هذا :

```
if Clipboard.HasFormat(cf_Bitmap) then  
    Image2.Picture.Bitmap.Assign(Clipboard)  
else
```

```
    Image2.Picture.Bitmap.Assign(nil);
```

ان استخدام خاصية الـ Picture يمكن الـ object من العمل مع بيانات الأيقونة ، و Bitmap والـ metafile . وهذا أيضاً يمكن الـ object من العمل مع أى TGraphic class للاضافة مثل الـ TJPEGImage من الـ TJPEGUnit . إن استدعاء الـ Assign لـ TBitmap يعد أكثر تحديداً ، ولكن أى من التقنيتين تعتبر مقبولة .

## Component objects و clipboard:

استدع الـ SetComponent method الخاص بالـ Clipboard لنسخ أى component الى الـ Clipboard . استدع الـ GetComponent method لـ Clipboard ، أما فى نفس التطبيق أو فى تطبيق آخر ، للصق الـ GetComponent . بصرى من الـ Clipboard .

داخلياً ، تستخدم الـ SetComponent والـ GetComponent الـ memory-streaming لنسخ component من والى الذاكرة . لاستعادة

component، بالإضافة إلى استدعاء ال GetComponent يجب ان تسجل ال component class مع نظام ال memory stream الخاص بـ Delphi. والمثال على انتقالات component clipboard. يوضح التقنيات الأساسية. اتبع هذه الخطوات لإنشاء تطبيقاً اختبارياً:

١- ابدأ تطبيقاً جديداً وأضف تعريف ال uses إلى قطاع ال implementation كما يلي:

```
implementation
uses Clipbrd;
```

٢- من Delphi Standard الخاصة بـ Delphi، أضف Button و ScrollBar (ال object الذى سيتم نسخه) على form. قم بإنشاء OnClick بالضغط مرتين على ال Button1، وأدخل العبارة التالية بين ال begin وال end لنسخ ال ScrollBar إلى clipboard عندما تضغط ال Button.

```
Clipboard.SetComponent(ScrollBar1);
```

٣- قم بتشغيل التطبيق بضغط F9، ثم اضغط الزر ينسخ ال ScrollBar1 إلى clipboard. اخرج من البرنامج لترجع إلى ال Delphi. احفظ التطبيق فى دليل مؤقت (اختر ال File|Save All واستخدم اسماء الملفات الافتراضية).

٤- ابدأ تطبيقاً جديداً، وأضف Button على form، وكما سبق، أضف تعريف uses لقطاع ال implementation.

```
implementation
uses Clipbrd;
```

٥- اضغط مرتين على ال Button1 لإنشاء OnClick، وأدخل العبارة التالية بين ال begin وال end.

```
if Clipboard.HasFormat(cf_Component) then
  Clipboard.GetComponent(Self, Self);
```

٦- قم بتحريك ال code editor إلى ال end النهائية فى ال unit module، وفوق هذا السطر مباشرة، أدخل عبارة البدء التالية. يجب أن تنتهى ال unit (متضمنة ال end النهائية والنقطة) كما هو موضح هنا:

initialization

RegisterClasses([TScrollBar]);

end.

٧- اضغط F9 لتشغيل التطبيق الثاني، واضغط الزر لنسخ الـ ScrollBar من الـ clipboard ولصقة على الـ form. (إذا لم تنجح هذه الطريقة، فقد تكون قمت بنسخ شيء آخر إلى الـ clipboard أثناء كتابة التطبيق الثاني. في هذه الحالة، أدخل التطبيق الأول المضغوط باستخدام الـ Windows Explorer، ثم من Delphi وقم بتشغيل التطبيق الثاني واضغط الزر).

إن وظيفة الـ GetComponent الخاصة بالـ Clipboard تتطلب إثنتين من الـ arguments التي تمثل الـ Parent والمالك لـ pasted object بالرغم من هذه الـ arguments غالباً ما تكون واحدة كتلك الموجودة في المثال السابق، فإنها تختلف عندما تلصق components في حاوية مثل الـ GroupBox. في هذه الحالة، تكون عادة هي المالك والـ GroupBox هو parent للـ pasted object، ويمكنك استخدام code مثل:

if Clipboard.HasFormat(cf\_Component) then

Clipboard.GetComponent(Self, GroupBox1);

**ملحوظة:** في الـ code السابق، أفترض أن العبارات موجودة في الـ form event handler أو لـ object في تلك الـ form، في هذه الحالة، يشير الـ Self إلى الـ form object.

Note

وتجدر الإشارة إلى أن الأمثلة السابقة تقدم تقنية برمجة الـ Object Pascal مفيدة ولكن نادراً ما تحتاج إليها. عند بدء التنفيذ لأي unit module عندما يبدأ التطبيق، أدخل عبارات بين كلمة الـ initializations الأساسية والـ end النهائية للـ unit. هذا يحل محل قطاع الـ begin-end initialization ذي الأسلوب القديم في الـ unit، والذي لا يزال Delphi يوفره. في الـ units الجديدة، استخدم الـ form الجديدة كما هو موضح هنا:

unit AnyUnit;

...8

initialization { same as begin }



// Statements to perform when application begins.

end.

قد تستخدم ال begin أو ال initialization ولكن الأخيرة تساعد ال compiler على إيجاد خطأ الإزدواج لل begin وال end الرئيسية فى مكان آخر فى ال module ، لذا تكون ال initialization هى المرشحة .

## Dynamic Data Exchange

إن ال Windows clipboard تعمل بشكل جيد ، ولكنها بدائية .

ولكن ال Dynamic Data Exchange (DDE) protocols موقع Internet Web الذى يتم الوصول إليه عبر ال modems . لنقل البيانات ، يبدأ تطبيقان ، معروفان بالعميل وال server ، محادثة التى تنساب من خلالها البيانات . بعد أن تقيم التطبيقات محادثة ، فيمكن أن تحدث إنتقالات ال DDE بصورة تلقائية ، أو يطلب أحد التطبيقات بيانات من الآخر .

إن محادثة ال DDE تشمل إثنين ، مرسل البيانات ويسمى ال server application ، ومستقبل البيانات ويسمى ال client application . إذا وجدت أن هذه المصطلحات محيرة ، فقط تذكر أن ال Server يقدم البيانات للعملاء . ولكن ، محادثة ال DDE هو طريق مزدوج ، وخلال المحادثة ، يمكن أن تنساب البيانات فى أى من الاتجاهين ، مما يشوش التميز بين العميل وال server .

إن ال DDE مناسبة لتحديث معلومات ضمن تطبيقات متعددة . على سبيل المثال ، مع ال DDE ، يمكنك كتابة برنامج يقوم بصورة تلقائية بتحديث صورة جرافيك عندما يدخل المستخدمون بيانات ورقة حسابات . أو ، يمكنك استخدام ال DDE لإنشاء تطبيقات متعددة تشارك فى المعلومات .

**Tip** **فكرة مفيدة:** تستطيع ال DDE أن تمنع تدمير البيانات من جانب

المستخدمين الغير خبراء . على سبيل المثال ، يمكنك كتابة تطبيق عميل لعرض النتائج لحسابات ورقة ال spreadsheet . ولكن ، لا يستطيع المستخدمون استخدام تطبيق العميل لتعديل صيغ ورقة الحسابات الأصلية .

### فهم الـ DDE components الخاصة بـ Delphi

استخدم الـ DDE components الأربع الخاصة بـ Delphi -  
DdeServerItem ، DdeServerConv ، DdeClientItem ، DdeClientConv  
لإنشاء تطبيقات عميل و server تستطيع أن تتصل باستخدام الـ DDE . والـ Coun  
تمثل الـ conversation والتي تمثل الصلة المقامة بين مهمتى الـ DDE . يمكنك أن  
تبدأ محادثة DDE عندما تصمم التطبيق ، أو يمكنك أن تربط العميل والـ server  
فى وقت التشغيل .

يعمل الـ DdeClientConv مثل الذى يستقبل استدعاء من حاسوب آخر .  
إن شكل محادثة العميل " يرفع السماعه " ليقوم رابطة مع النظام البعيد الذى أنشأ  
الاستدعاء . إن أغلب تطبيقات الـ DDE تحتاج إلى DdeClientConv واحد  
فقط ، بالرغم من أنه بإمكانك استخدام object متعددة لإقامة أكثر من مسار اتصال  
واحد .

ويمثل الـ DdeClientItem البيانات التى تأتى عبر الـ modem على مسار  
المحادثة . ان التطبيق الذى يستقبل أنواع متعددة من البيانات قد يكون له العديد من  
الـ DdeClientItem المرتبطة بنفس الـ DdeClientConv . ولكن ، تطبيق العميل  
يحتاج إلى DdeClientConv واحد فقط و DdeClientItem مرتبط واحد .

فى تطبيق Server بعبارة أخرى ، مرسل البيانات استخدم الـ  
DdeClientConv لبدء محادثة . انك عادة ما تحتاج إلى DdeClientConv واحد  
فقط لكل تطبيق ، ولكن يمكنك استخدام عدة objects لإقامة محادثات متعددة .  
ادخل واحداً أو أكثر من الـ DdeClientItem لتمثل البيانات التى يتم إرسالها  
للعملاء . ان أغلب تطبيقات الـ DdeServer تحتاج إلى DdeClientConv واحد  
فقط DdeClientItem واحد مرتبط .

**فكرة مفيدة:** لإرسال واستقبال معلومات DDE فى نفس التطبيق ،  
أضف DdeClient و DdeServer على نافذة form المشروع . وهذا  
أيضاً مفيد فى اختبار مفاهيم الـ DDE .

Tip



## اختبار مصطلحات محادثة ال DDE ،

هناك ثلاث مصطلحات تعرف محادثة ال DDE وعناصر البيانات التي تناسب بين الإثنين أو أكثر من التطبيقات . وهذه المصطلحات هي :

\* Service : يُعرف ال Server ، عادة باسم الملف التنفيذي الخاص به ناقص امتداد ال .exe . فى بعض الحالات ، رغم ذلك ، يعتبر ال DDE Service تعريف مختلف . فى تطبيق عميل ال Delphi ، يكون ال Service دائماً هو اسم ملف ال Service ناقص ال .exe .

\* Topic : يُعرف unit من البيانات مثل اسم الملف ، أو Caption نافذة . لا يوجد قواعد محددة عما تعينه ال Topic . فى تطبيقات Delphi ، رغم ذلك ، فان ال Topic يكون إما Caption النافذة الرئيسية لتطبيق المساعد ، أو اسم ال DdeServerItem فى المساعد .

\* Item : يُعرف عنصراً من بيانات النص المرسلة بواسطة Server إلى تطبيق عميل . يمكن أن يكون النص أى طول ، ولكن أغلب انتقالات ال DDE تكون صغيرة . وللقوالب الكبرى من النص ، وبالأخص تلك المنظمة سطور ، تمثل ال DDE components الخاصة بـ Delphi Items كقوائم TStrings . string

## اقامة محادثات بين العميل وال Server ،

كمثال على استخدام ال DDE ، فإنك تنشئ فى هذا الفصل تطبيقات عميل Server . ببساطة تنقل النص بين الإثنين من ال Edit controls . إن كتابة بيانات فى نافذة ال Edit للمساعد تؤدي إلى تحديث ال Edit الخاص بالعمل بصورة تلقائية . ولكن ، الكتابة فى نافذة ال Edit للعميل لا تؤدي إلى تحديث ال Server . وهذا يعد مثلاً على محادثة ال DDE ذات الاتجاه الواحد .

**على القرص المدمج:** إن قوائم البرمجة فى هذا الفصل والفصول القليلة القادمة موجودة فى الدليل الفرعى Source\Dde1 على القرص المدمج .



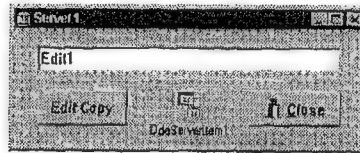
## كتابة ال server :

إذا لم يكن لديك تطبيق DDE server، فمن الأفضل دائماً أن تطور server قبل كتابة العميل . كحد أدنى، يطلب تطبيق ال server DdeServerItem object وبعض البيانات لإرسالها للعملاء . اتبع هذه الخطوات لإنشاء مساعد بسيط يرسل نص Edit لتطبيق العميل . لقد استخدمت الاسماء الافتراضية لل components، والتي تساعد على توضيح العلاقات بين ال objects وال DDE.

١- قم بإنشاء دليل قرص جديد لعمل ملفات المشروع . ابدأ تطبيقاً جديداً وحدد خاصية ال Name لل Form1 بـ ServerForm . حدد خاصية ال Caption لل Form1 بـ Server1 .

٢- اختر File|Save All واضغط المشروع في الدليل الجديد . اجعل اسم ال unit module بـ Server.pas والمشروع Server1.dpr .

٣- أدخل DdeServerItem من ال System VCL على ال form . يرسل ال server بيانات من خلال هذا ال object . وكذلك، أضف ثلاثة objects : ال Edit، ال Button، وال BitBtn . حدد ال Caption الخاص بال Button1 بـ Edit Copy . قم بتغيير خاصية ال Kind لل BitBtn1 لتصبح bkClose . رتب وحدد حجم ال form لتشبه الشكل ١٦-٢ .



شكل ١٦-٢، نافذة ال form ال Server1 في Delphi.

٤- قم بإنشاء ال OnChange الخاص بال Edit1 . أدخل العبارة الموجودة في القائمة ١٦-٥، ال Edit1Change procedure، لتعيين خاصية ال Text التابعة لل Edit1 إلى ال Text الخاص بال DdeServerItem1 . عندما يتغير نص ال Edit1، فهذا يرسل النص إلى أى تطبيقات عميل تكون قد أقامت رابطة مع ال Server.

٥- قم بإنشاء الـ Button1 OnClick . أدخل العبارة من القائمة ١٦-٥ ، الـ CopyToClipboard الـ method لإستدعاء الـ Button1Click procedure الخاص بالـ DdeServerItem . هذا يخبر تطبيقات العميل عن أسماء الـ Topic والـ Service الخاصة بالـ server . إنك تستخدم هذه المعلومة فى كتابة تطبيق عميل .

#### القائمة ١٦-٥ ، Dde1\Server.pas

```
unit Server;

interface

uses
  Windows, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, DdeMan, Buttons;

type
  TServerForm = class(TForm)
    DdeServerItem1: TDdeServerItem;
    Edit1: TEdit;
    Button1: TButton;
    BitBtn1: TBitBtn;
    procedure Edit1Change(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  ServerForm: TServerForm;

implementation

{$R *.DFM}
```

```

procedure TServerForm.Edit1Change(Sender: TObject);
begin
    DdeServerItem1.Text := Edit1.Text;
end;

procedure TServerForm.Button1Click(Sender: TObject);
begin
    DdeServerItem1.CopyToClipboard;
end;

end.

```

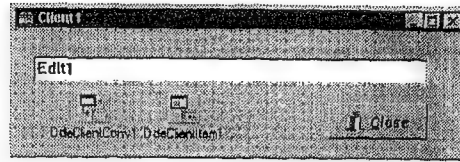
### كتابة العميل:

إن كل تطبيق server يحتاج إلى واحد أو أكثر من العملاء ليشارك في محادثة وبت البيانات. إتبع هذه الخطوات لإنشاء تطبيق عميل للحصول على بنود نص من (server) الذي تم إنشاؤه في الفصل السابق. في الخطوة رقم (٢)، تأكد من أن تحفظ ملفات المشروع في نفس دليل الـ (server).

١- إبدأ تطبيقاً جديداً. إجعل اسم الـ form بـ ClientForm، وغير الـ Caption الخاص بها ليصبح Client1.

٢- استخدم الـ FileSave لحفظ ملفات المشروع في نفس الدليل الذي يحمل مشروع الـ Server1. إجعل إسم الـ unit module بـ Client.pas والمشروع Client1.dpr.

٣- أضف DdeClientConv من الـ System VCL على الـ form. وكذلك، أضف الـ DdeClientItem. على العكس الـ (server)، تحتاج تطبيقات العميل إلى object محادثة وعنصر لإقامة محادثة (server) ولاستقبال عناصر البيانات. لتوفير مكان لعرض البيانات التي تم استقبالها، أضف Edit معيارى على الـ form. وكذلك، أضف BitBtn وقم بتغيير خاصية الـ Kind له لتصبح bkClose. إجعل form العميل مشابهاً لشكل ١٦-٣. قد تريد تحريك الـ form إلى أسفل حتى لا تغطي بالضبط نافذة الـ (server).



شكل (١٦-٣) نافذة Form ال Client1

- ٤- اختر ال DdeClientItem1 (الأيقونة التى ليس بها اسمهم) وحدد خاصية ال DdeConv بـ DdeClientConv1. هذا يقيم العلاقة بين المحادثة وشكل العنصر، ويخبر ال DdeClientItem1 من أين يتلقى بياناته.
- ٥- استخدم ال Windows Explorer لتشغيل تطبيق ال Server. (إنك تحتاج الى أن تقوم بتشغيل كلا التطبيقين، وهو مالا تستطيع أن تفعله من داخل Delphi). اضغط زر ال Edit Copy لتنسخ أسماء ال Topic وال Service الخاصة بال (Server) الى ال Clipboard. يمكنك أن تترك ال Server1 عاملاً الى نهاية هذه الخطوات.
- ٦- ارجع الى Delphi. اختر ال DdeClientConv1 (الأيقونة ذات الأسهم)، واضغط الزر البيضاوى الواقع بعد خصائص ال DdeService أو ال DdeTopic (لايهم أى منهما). اضغط زر ال Paste Link الخاص بال DDE Info dialog الناتج. يجب أن ترى ال SERVER1 Strings وServer1 فى نافذتى التحرير. إذا لم تراهما، انتقل الى ال Server1، اضغط Edit Copy، ثم انتقل على الفور مرة أخرى الى Delphi وحاول ثانية. إذا كنت تعرف ال Service String وال Topic الخاصة بال (Server)، يمكنك إدخالها فى ال dialog، ولكن هذا يؤدي الى محاولة لتنفيذ (Server) ويجب أن يكون كلا ال Strings صالحان أو يرفضهما Delphi. اضغط OK لإغلاق ال DDE Info dialog.
- ٧- قم بإنشاء ال OnCreate الخاص بال Form، وادخل البرمجة من القائمة (١٦-٦) لتعيين اسم ال DdeServerItem1 الخاص بال (Server) لخاصية ال DdeItem فى ال DdeClientItem1 الخاص بالعمل. لقد حدد العمل الآن ال Strings (SERVER1) Service، وال (Server1) Topic وال Item (DdeServerItem1) التى تتطلبها محادثة ال DDE. يمكنك أن

## الباب السادس عشر : التطوير مع الـ Clipboard، الـ DDE، والـ OLE

تحدد الـ Server والـ Topic في وقت التصميم ، ولكنك عادة ماتعين الـ Item في وقت التشغيل لأن، إذا كانت خاصية الـ ConnectMode التابعة للـ DdeClientConv محددة بـ ddeAutomatic (القيم الافتراضية) ، فإن خصائص الـ DdeServerItem1 والـ Text في الـ DdeClientItem1 تكون عرضة للتغيير عبر الرابطة المقامة . وبسبب طبيعة الـ DD التفاعلية، فهذه الحقيقة تكون صادقة حتى في وقت التصميم .

٨- أخيراً، أنشئ الـ DdeClientItem1 OnChange ، الذي يحدث عندما تأتي معلومات جديدة من الـ (Server) . ادخل العبارة من القائمة (١٦-٦) لتعيين الـ Text الخاص بالـ DdeClientItem1 (ممثل البيانات التي تم استقبالها) الى خاصية الـ Text التابعة للـ Edit1 .

٩- أحفظ المشروع . اضغط F9 لتشغيل الـ Client1 . إذا لم يكن الـ Server1 في حالة تشغيل بالفعل، فإن هذا يجعل الـ Server يبدأ أيضاً (Server) . اجعل نافذتي الـ Client1 والـ Server1 مرئيتين ادخل نصاً في نافذة الـ Edit للـ Server1 . يجب أن ترى كتابتك في الـ ClientEdit1 . ولكن، النص الذي يدخل في الـ Client لا ينعكس مرة أخرى الى الـ (Server) لأن هذه هي محادثة ذات اتجاه واحد .

**ملحوظة:** إذا واجهت صعوبات في إقامة محادثة DDE، إما أن تترك Delphi أو تبدأ مشروعاً جديداً وقم بتشغيل تطبيقات (Server) والعمل باستخدام الـ Windows Explorer ، أو الـ Taskbar . لأسباب غير معروفة، يبدو أن تشغيل تطبيق عميل في نمط الـ Debug الخاص بـ Delphi لا يقيم دائماً محادثة بطريقة سليمة مع الـ (Server) . كذلك، حاول تشغيل (Server) قبل بدء العميل، والذي قد يكون ضرورياً إذا كان الملف التنفيذي للـ (Server) ليس في دليل PATH نظامي .

Note

### القائمة (١٦-٦) DdelClient.pas

unit Client;

interface



```

uses
  Windows, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, DdeMan, StdCtrls, Buttons;

type
  TClientForm = class(TForm)
    DdeClientItem1: TDdeClientItem;
    DdeClientConv1: TDdeClientConv;
    Edit1: TEdit;
    BitBtn1: TBitBtn;
    procedure FormCreate(Sender: TObject);
    procedure DdeClientItem1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  ClientForm: TClientForm;

implementation
  {$R *.DFM}

  procedure TClientForm.FormCreate(Sender: TObject);
  begin
    DdeClientItem1.DdeItem := 'DdeServerItem1';
  end;

  procedure TClientForm.DdeClientItem1Change(Sender:
  TObject);
  begin
    Edit1.Text := DdeClientItem1.Text;
  end;

end.

```

### إقامة محادثات في وقت التشغيل؛

يوضح الفصل السابق كيفية إقامة محادثة DDE بواسطة تحديد Service Strings و Topic في الـ Object Inspector. يمكنك أيضاً أن تبدأ محادثة DDE في وقت التشغيل، والذي قد يكون أكثر ملائمة، خاصة إذا كان برنامجك يختلف مع كثير من (Servers) حول مسارات المحادثة المتعددة.

إن إقامة محادثة الـ DDE في وقت التشغيل هي مهمة تطبيق العميل. إنك تكتب تطبيق (Server) بشكل لا يختلف عما كتبه في المثال السابق. يقوم العميل بإنشاء أسماء الـ Service والـ Topic للـ DDE ويحدد اسم عنصر البيانات الذي يتلقاه من (Server). ولأن البرنامج يؤدي هذه الأعمال في الـ Code، فإنه يتعين عليه أيضاً أن يستدعي واحداً أو اثنين من الـ Methods ليربطهما بالـ (Server).

**على القرص المدمج:** على سبيل المثال، يمكنك تحويل تطبيق الـ Client1 السابق لبدء محادثة في وقت التشغيل بحذف الـ Strings DdeService والـ DdeTopic من الـ DdeClientConv1. اختياريًا، حدد الـ ConnectMode لـ ddeManual، ثم قم بتعديل الـ event handler Form ليتوافق مع القائمة (١٦-٧) (القائمة مأخوذة من تطبيق عميل معدل في دليل الـ Source\Dde2 على القرص المدمج).



### القائمة (١٦-٧) OnCreate المعدل يوضح كيفية إقامة

#### محادثة DDE في وقت التشغيل

```
procedure TClientForm.FormCreate(Sender: TObject);
begin
  if DdeClientConv1.SetLink('SERVER2', 'Server2') then
    begin
      DdeClientItem1.DdeItem := 'DdeServerItem1';
      if DdeClientConv1.ConnectMode = ddeManual then
        DdeClientConv1.OpenLink;
    end;
end;
```

OnCreate المعدل يستدعى الـ DdeClientConv1 SetLink . يقوم الـ Strings بإنشاء خصائص الـ DdeService والـ DdeTopic على التوالى - لا تقوم بتعيين String الى هذه الخصائص ، استدع دائماً الـ SetLink لإنشاءهم . تقوم الـ SetLink بإدخال True إذا كانت قادرة على ربط الـ Server المحدد . وكذلك ، قم بتعيين اسم بند لخاصية الـ DdeItem فى الـ DdeClientItem . أخيراً ، إذا كان الـ ConnectMode التابع للـ DdeClientConv1 محدد بـ ddeManual ، استدع الـ OpenLink لتبدأ المحادثة . إذا كان الـ ConnectMode محدد بـ ddeAutomatic ، فإن الـ SetLink يستدعى الـ OpenLink بصورة تلقائية .

### تحديد الـ Service والـ Topic :

لتحديد الـ Service والـ Topic لتطبيق DDE server ، اتبع هذه الخطوات :

- ١ - قم بتشغيل التطبيق .
- ٢ - اختر بعض البيانات .
- ٣ - اختر Edit/Copy . (جرب هذا مع الـ Microsoft Excel) .
- ٤ - انتقل الى Delphi .
- ٥ - أضف DdeClientConv على form .
- ٦ - اضغط الزر البيضاوى بعد خاصية الـ DdeService .
- ٧ - اضغط Paste Link فى DDE Info dialog .

هذا يجب أن يعرض الـ Service string والـ Topic الذى قام (Server) بلصقها الى الـ Clipboard . ولكن ، هذه التقنية لاتخبرك بأسماء عناصر البيانات ، التى تحتاجها لإتمام تطبيق العميل . كن مستعداً للقيام ببعض أعمال المخبرين لتحديد أسماء بنود (Server) . إن أسماء البنود تحدد بشكل نموذجى عناوين البيانات ، أو المواقع الأخرى . على سبيل المثال ، إن عناصر تطبيق ورقة الحسابات (Spreadsheet) تكون غالباً صفوف خلايا ، عناصر قاعدة البيانات قد تكون أسماء حقول مسجل .

### تلقى بيانات من ال DDE server ،

إن تطبيق عميل ال DDE يتلقى بيانات من (server) من خلال ال DdeClientItem . قم بتنفيذ OnChange التابع لهذا ال Object لتستجيب للبيانات الجديدة التي تأتي من (Server) . على سبيل المثال ، يقوم OnChange في القائمة (١٦-٦) بتعيين خاصية ال Text من ال DdeClientItem1 الى ال Text التابعة لل Edit .

وبالنسبة ال blocks الطويلة من بيانات النص ، خاصة عندما تنظم في صفوف ، استخدم خاصية ال DdeClientItem Lines ، وهو object من نوع ال TStrings . على سبيل المثال ، لتتلقى بند نص متعدد الأسطر في محادثة DDE ، استخدم Code مثل التالية ، التي تعين النص الذي تم تلقيه الى خاصية ال Lines التابعة لل Memo :

```
with ddeClientItem1 do
  Memo1.Lines := ddeClientItem1.Lines;
```

إن أغلب تطبيقات ال DDE تتلقى البيانات بصورة تلقائية من خلال ال DdeClientItem OnChange . ولكن ، تطبيق العميل يمكن أيضاً أن يطلب بيانات محددة من (Server) - خلية معينة في ورقة حسابات Spreadsheet - مثلاً . لطلب بند معين من البيانات ، استدع ال DdeClientConv RequestData function ، والتي ترجع PChar pointer الى null-terminated . على سبيل المثال ، هذه الجزئية من ال Code تطلب عنصر DdeServerItem1 وتعين ال null-terminated الناتج للتعليق الخاص بال Label . استدع دائماً ال StrDispose لتتخلص من الذاكرة التي تخصصها ال RequestData للبيانات الواردة .

```
var
  P: PChar;
begin
  P := DdeClientConv.RequestData('DdeServerItem1');
  if P <> nil then
    try
      Label1.Caption := StrPas(P);
    finally
```

```
StrDispose(P);
end;
end;
```

### ارسال بيانات ال Server ال DDE server :

إن إرسال بيانات الى عميل ال DDE يعتبر غالباً أمراً بسيطاً من تعيين String لخصائص ال Lines وال Text التابعة لل DdeServerItem. هذا يؤدي الى إرسال البيانات بصورة تلقائية الى أى عملاء يقيم ال (Server) محادثة معهم. وبالتبادل، يستطيع تطبيق العميل أن يعكس الأدوار ويرسل بيانات الى (Server). لفعل هذا، استدع ال PokeData methods (Strings فردية) أو PokeDataLine لل DdeClientConv الخاص بالعميل. وكل Function ترجع بـ True إذا كانت ناجحة. قم بتمرير اثنين من ال string arguments الى ال PokeData : اسم ال DdeServerItem الذى تريد أن ترسل له البيانات ، والنص الذى تريد إرساله :

```
with DdeClientConv1 do
```

```
  if not PokeData('DdeServerItem1', 'I'd rather go sailing')
  then
    ShowMessage('Poke failed');
```

استخدم ال PokeDataLines بنفس الطريقة ، لكن قم بتمرير ال TStrings أو TStringList على أنه ال argument الثانية. إن ال Pokin يعمل فقط إذا تقبل (server) ال Poked data. فى تطبيقات ال Delphi، يجب أيضاً أن تنشئ ال DdeServerItem OnPokeData، الذى يتم اطلاقه عندما يحاول العميل أن يدفع البيانات الى ال server المتعاون معه. فى ال event handler، تكون البيانات المتاحة من خلال خصائص ال Lines وال Text التابعة لل DdeServerItem.

### استخدام تعليمات ال macros الخاصة بال DDE:

إذا كان المساعد يقدم تعليمات macros، فيستطيع تطبيق العميل أن يرسل أوامر macros لأداء العمليات فى ال Server. على سبيل المثال، يستطيع تطبيق العميل ال DDE الخاص بـ Delphi أن يستخدم هذه التقنية لإرسال وتنفيذ أمر macro لوثيقة ال Microsoft Word.

## الباب السادس عشر : التطوير مع الـ Clipboard، الـ DDE، والـ OLE

في تطبيقات الـ Delphi الخالصة، يرجع تحديد معنى الـ Macro إليك أنت. ولكن إذا لم يكن الـ Server الخاص بك يؤيد الـ Macro، فيمكنك استخدام الـ Components لإرسال واستقبال أوامر الـ Macro. لفعل هذا، يقوم الـ DdeServerConv بتنفيذ الـ OnExecutemacro. عندما يقيححدث هذا الـ event، يجب أن يقوم الـ Server الخاص بك بتنفيذ أوامر الـ Macro في خاصية الـ DdeSererItem Lines. (بالرغم من أنه بإمكانك استخدام خاصية الـ Text أيضاً، فإنك غالباً ما تستخدم الـ Lines، لأن الـ Macro غالباً ما يتكون من سطور متعددة).

وعلى الوجه الآخر للعملة وهو ما يخص العميل استدع الـ ExecuteMacro أو الـ ExecuteMacroLines للـ DdeClientConv. أنك عادةً تستخدم الـ ExcuteMacroLines لإرسال أمر Macro متعدد الأسطر إلى الـ Server ليقيم بتنفيذه. إذا كان هذا الأمر موجود في الـ TStringList يدعى الـ Macro، استخدم هذا الـ Code لإرساله إلى الـ Server :

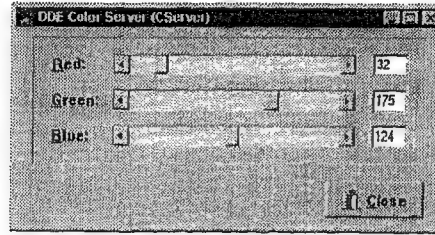
```
with DdeClientConv1 do
  if not ExecuteMacroLines(Macro, True) then
    ShowMessage("Macro failed");
```

إن الـ ExcuteMacro والـ ExcuteMacroLines ترجع بـ True إذا كانا ناجحين. إن الـ argument الأولى للـ ExcuteMacro هي الـ PChar تشير إلى الـ String، وهي الـ TStringList أو الـ TStringList. وتكون الـ argument الثانية True إذا أردت للبرنامج أن ينتظر الـ Server حتى ينفذ أمر الـ macro؛ حدد الـ argument الثانية بـ False لتمكين العميل من الاستمرار بينما الـ Server يقوم به مع الـ macro الذي تم إرساله.

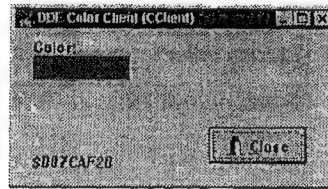
### اختبار برنامج DDE :

على القرص المدمج: كمثال أخير على محادثات الـ DDE، يحتوي دليل الـ DdeColor على القرص المدمج على تطبيقين، الـ CServer، والـ CClient، يرسلان ويستقبلان معلومات لون عبر محادثة الـ DDE. يوضح شكل (١٦-٤) عرض الـ CServer. يوضح شكل (١٦-٥) نافذة الـ CClient.





شكل (٤-١٦) يرسل الـ CServer معلومات لون الى أى من تطبيقات العميل التى تقيم مجادثة DDE مع الـ CServer



شكل (٥-١٦) يبدأ الـ CClient مجادثة مع الـ Server ويتلقى بصورة آلية كعلاومات لون عبر الـ Link

لتشغيل البرامج، اتبع الخطوات التالية:

- ١- انتقل الى نسخة من دليل الـ Source\DdeColor على القرص المدمج.
- ٢- قم بتحميل ملف مشروع الـ CServer.dpr فى Delphi.
- ٣- اضغط Ctrl+F9 للـ compile، ولكن ليس لتشغيل البرنامج.
- ٤- قم بتحميل الـ CClient.dpr من نفس الدليل، ومرة أخرى اضغط Ctrl+F9 للـ compile.
- ٥- اغلق كل الملفات فى Delphi.

٦- باستخدام Windows Explorer، قم بتشغيل الـ CClient.exe. هذا أيضاً يؤدى إلى بدء CServer، إلا إذا كان عاملاً بالفعل (قم بتصغير نوافذ أخرى إذا أصبحت نافذة البرنامج غير مرئية). استخدم scroll bar بالـ server لتحديث اللون فى تطبيق العميل.

**ملحوظة:** للحصول على أفضل النتائج، قم دائماً بتشغيل تطبيقات الـ DDE باستخدام الـ Explorer. يمكنك تشغيل برامج DDE من داخل

Note

## الباب السادس عشر : التطوير مع الـ Clipboard، الـ DDE، والـ OLE

DDE، ولكنني واجهت مشاكل في إقامة روابط الاتصال في بعض الأحيان. قبل إهدار وقتك في إزالة الأخطاء من برامجك، حاول تشغيلها من خارج Delphi. (إن الوضع يجب أن يكون قد تحسن كثيراً في Delphi4 عنه في Delphi2، وقد لا تواجه هذه المشكلات).

إن المساعد المثالي له ثلاثة scroll bars التي يمكنك تعديلها لإنشاء قيمة لون مكونة من قيم الـ Byte الأزرق والأخضر والأحمر من صفر إلى ٢٥٥. الـ server يجعل اللون المختار متاحاً إلى أى تطبيقات عميل تقيم محادثات مع server. ويتلقى العميل المثال بصورة تلقائية قيم اللون الخاصة لـ server، والتي يمثلها البرنامج في Edit ثم ابطاله. ويعرض Label قيمة اللون في كسر عشري. إن الألوان التي تتوافق مع الألوان المعيارية لـ Delphi، مثل الـ cl\_White والـ cl\_Fuschia، يتم عرضها بالاسم.

والإختلاف الملحوظ في تطبيق الـ SServer هو استخدام الـ DdeServerConv لبدء محادثة. استخدم هذا الـ Object لتحديد DDE Topic يكون غير مساوياً للتعليق الخاص بنافذة Server. في الـ CServer، مثلاً، يسمى الـ DdeServerConv الـ ColorServer. والـ DdeServerItem المقابل يحدد الـ ColorServer في خاصية الـ ServerConv التابعة له - هذا يخبر عنصراً Server كيف يقوم بإرسال بيانات خلال محادثة. في تطبيق الـ CClient، تكون خاصية الـ DdeTopic للـ DdeClientConv1 محددة أيضاً لـ ColorServer.

**على القرص المدمج:** توضح القائمة (١٦-٨) كيف يتلقى تطبيق الـ CClient من الـ Server عنصر نص يمثل قيمة لون بالكسر العشري. هذه الجزئية هي من تطبيق الـ CClient في دليل الـ Source\DdeColor على القرص المدمج. ولكن تحويل هذا النص إلى لون ثنائي بطريقة آمنة أمر يتطلب برمجة حذرة.



**القائمة (١٦-٨) تطبيق الـ CClient يتلقى بند نص من الـ Server**

```
procedure TMainForm.DdeClientItem1Change(Sender: TObject);
var
  S: String;
```



```

C: TColor;
begin
  with DdeClientItem1 do
    begin
      if Length(Text) = 0 then
        S := '$0' { Default string }
      else
        S := Text; { String from server }
      try
        { Delete trailing blanks }
        while S[Length(S)] = asciiBlank do
          System.Delete(S, Length(S), 1);
        C := StringToColor(S); { Convert to
color }
        ColorValueLabel.Caption := S; { Assign to
label }
        Edit1.Color := C; { Show color }
      except
        ShowMessage('Bad color format from server');
      end;
    end;
  end;
end;

```

قد يكون النص الذي تم استقباله بلا طول ، وفي هذه الحالة يحدد الـ Object DdeClientItem1 OnChange للمتغير النتيجة الافتراضية (\$0) (في الـ Pascal، تكون قيم الـ strings العشرية مسبقة بعلامة الدولار). وإن لم يكن كذلك ، يعين البرنامج الـ Text الآتي إلى الـ strings. قد يكون هذا المتغير متبوعاً بفراغات ، وهو الأمر الذي قد يجعل الـ StringToColor function لتوليد exception. لذلك ، تقوم الـ while loop بإزالة أية فراغات لاحقة. لاحظ أن البرنامج يستدعي Delete مسبقة لـ System. وهذا ضرورياً لأن الـ Delete هي أيضاً method في Delphi components عديدة. لاستدعاء الـ string Delete method في نظام ، فانك غالباً تستخدم الصيغة System.Delete كما هو موضح هنا .

**ملحوظة:** بعض Servers قد يرسلون control codes فى بيانات النصوص الخاصة بهم. حدد خاصية ال FormatChars بـ True لل DdeClientConv الخاص بتطبيق العميل. هذا يؤدى الى فصل control code المسافات الخلفية (#8)، ال (#9) tab، ال (#10) line feed، وال (#13) carriage return عن النص الذى يتم استقباله.

Note

## ال Object Linking and Embedding (OLE) :

إن تقنية تشارك البيانات الأكثر تعقيداً والمتاحة فى ال Windows تعرف بال Object Linking and Embedding. أو ال OLE. والميزة الرئيسية التى تقدمها ال OLE هى التحول من نظرة مختصة بالتطبيق للحاسوب الى أخرى تتركز على الوثائق. إن مستخدمى الحاسوب يعملون بالبيانات، وتركيز الإهتمام على الوثائق ك Objects تبدو أكثر طبيعية لكثير من المستخدمين. مع ال OLE، يستطيع المستخدمين أيضاً أن يجمعوا المعلومات بطرق غير متوقعة. على سبيل المثال، يمكن أن تحتوى وثيقة برنامج معالجة كلمات على صور جرافيكية ثم إنشاءها بواسطة برنامج غير معروف لمؤلف البرنامج المعالج للكلمات. وال OLE تجعل من الممكن على المستخدمين، وليس فقط على مصممي البرامج، أن ينشئوا أنواعاً جديدة من الوثائق غير مفيدة بـ form معينة لتطبيق واحد.

واستخدام آخر لل OLE هو الوصول الى Object فى تطبيقات أخرى Microsoft word، مثلاً، وهذا ممكن لأن ال OLE 2 مؤسس على ال Comon Object Model، أو ال COM. وهذه ال objects توفر واجهة تطبيق يستطيع أى تطبيق أن يستخدمها للوصول الى خصائص ال Objects، بغض النظر عن اللغة التى استخدمتها لإنشائها. COM Objects، ولذلك، فإن ال Objects ال OLE، تم إنشاءها من طراز ثنائى يوفر Delphi إليه الوصول التام.

وكما هو الحال مع ال DDE، فإن ال OLE، فإن ال OLE يتطلب تطبيق متعاونين - عميل (Server). ويعرف تطبيق العميل بحاوية ال OLE. ويوفر ال Server أوامر لتحرير أنواع معينة من البيانات. تتلقى الحاوية ال objects من ال OLE server، إما مباشرة، بواسطة ال Windows Clipboard، أو بسحب واسقاط ملفات من ال Windows Explorer. على سبيل المثال، ك OLE server، فإن

محرر ال Form موجة سمعية يمكن أن يوفر أوامر لإنشاء واثائق صوتية . تستطيع حاوية ال OLE أن تخزنها مع النص الذى يصف الصوت . توفر الحاوية أوامر عامة لإنشاء واثائق صوتية نصية جديدة ، ولكن ال OLE server يوفر الأوامر لإنشاء ، تحرير ، ورؤية أو تشغيل أجزاء فردية من البيانات .

### إنشاء ال OLE objects :

يستطيع التطبيق أن ينشئ OLE object من أى نوع مسجل من خلال ال Windows registry . على سبيل المثال ، يمكنك إنشاء OLE object فى ال Microsoft Word ، ثم تستخدم هذا ال object لاستدعاء methods تكتب نصاً ، تختار Fonts ، تطبع صفحات ، وتحفظ واثائق فى ملفات قرص . تستطيع تطبيقاتك أن تودى أعمالاً مشابهة مع تطبيقات ال OLE server - ال Microsoft Excel ، مثلاً .

لإنشاء OLE object ، استدع Creat eOleObject function فى Delphi ComObj unit أضف تعريف uses لإدخال هذه ال unit فى unit ، غالباً فى قطاع ال implementation :

```
implementation
uses ComObj;
```

**تحذير:** إن العديد من المراجع المطبوعة ومراجع ال online المتوفرة مع Delphi تخبرك خطأ أن تستخدم ال OleAuto unit . لاستدعاء ال CreateOleObject ، يجب أن تستخدم ComObj كما هو موضح هنا .



ال ComObj unit يعرف ال CreateOleObject وال parameters

الخاصة بها كما يلى :

```
function CreateOleObject(const ClassName: string): IDispatch;
```

● **ClassName** : هذا هو اسم ال class المسجلة - وهو Word.Basic أو Word.Document ، على سبيل المثال - الذى تريد أن تنشئ منه ال OLE object .

● **IDispatch** - إذا كانت ناجحة ، تقوم ال CreateOleObject بإدخال object من نوع ال IDispatch ، وهو تعريف واجهة تطبيق مؤسس على ال

## الباب السادس عشر : التطوير مع ال Clipboard، ال DDE، وال OLE

لـ IUnknown، كلاهما يتم تعريفهما في ال System.pas unit لـ Delphi. إن ال IDispatch يتم تعيينه الى Variant، والذي يتم استخدامه حينئذ لعمل استدعاءات لـ methods وللوصول الى خصائص في ال OLE.

إن اثنين من البرامج النموذجية يوضحان كيفية استخدام ال CreateOleObject. في كل برنامج، تقوم الأوامر بإنشاء وثيقة Microsoft Word، وادخال نص في الوثيقة، وطباعتها وحفظها في ملف doc. على القرص. وهذه الخطوات تحدد التقنيات الأساسية المطلوبة لإنشاء واستخدام OLE Objects- قد تستخدم برمجة مشابهة لإخراج نص تطبيق في ال Form وثيقة تكون مألوفة للمستخدم. على سبيل المثال، يستطيع نظام قاعدة البيانات أن يعد تقارير ويحفظهم في وثائق Microsoft Word أو Excel، كل ذلك دون أن يطلب من المستخدمين أن يفتحوا تلك التطبيقات.

**ملحوظة:** إن السبب في أن لهذا الفصل اثنين من البرامج المثالية هو أن ال Microsoft Word قد استبدل ال Word Basic- وهي لغة تعليمات ال macro القديمة للـ Word وال Excel- بالـ Visual Basic في ال Microsoft Office 97. هذا يعني أن غالبية الأوامر، وال macro (أو التعليمات)، وال OLE code لم تعد تعمل بالـ Word 97. إن أسس إنشاء ال OLE objects لم تتغير، ولكن التفاصيل أصبحت مختلفة تماماً. إذا كان لديك ال Windows 95 Word أو نسخة أقدم من هذه، فانظر أوامر ال Word Basic باستخدام ال online help للبرنامج. إذا كان لديك ال Word 97 الحديثة، إبدأ ال Visual Basic Editor باستخدام أمر ال ToolsMacro...، ثم اضغط زر ال Object Browser لتتمكن من الوصول الى ال online help قابل للبحث للـ Visual Basic.

### ال Word 95 OLE object والنسخ الاقدم:

**على القرص المدمج:** إذا كان لديك Word 95 أو نسخة أقدم، قم بتحميل ملف مشروع ال OleWord1.dpr من على القرص المدمج في



دليل OleWord1\Source الى Delphi . (استخدم OleWord2 إذا كان لديك Word 97 أو ما هو أحدث) يبدأ ال Word الآن إذا أردت أن ترى الوثيقة التي تم إنشاءها . إذا كان ال Word لا يعمل ، فإنه يكون قد تم تنفيذه ك Server صامت . فى هذه الحالة ، تكون الوثيقة مازالت تنشأ وتطبع ، ولكن يجب أن يكون ال Word يعمل حتى ترى نص الوثيقة وهى يتم إدخالها . أيضاً ، قم بتشغيل طابعتك - إذا لم يكن لديك واحدة فسوف تتلقى رسالة خطأ من النظام العامل (إن هذه غير ضارة ، ولكن بإمكانك أن تحذف عبارة V.FilePrint; لتجنبها) .

اضغط F9 لتشغيل البرنامج ، ثم اضغط زر إنشاء وثيقة Word ، ادخل النص "Hello form Delphi" ، اطبع الصفحة ، واحفظ الوثيقة على أنها C:\Hold.doc. (هذا هو الاسم الذى استخدمته للملفات المؤقتة) توضح القائمة (١٦-٩) ال Source code للبرنامج .

#### القائمة (١٦-٩) OleWord1\Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TMainForm = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

var
  MainForm: TMainForm;
```

implementation

uses ComObj; // Declares the CreateOleObject function

{ \$R \*.DFM }

{ This procedure works with Microsoft Word 95 and earlier versions. It does \*not\* work with the newer Word 97, which now uses Visual Basic in place of Word Basic. }

procedure TMainForm.Button1Click(Sender: TObject);

var

V: Variant;

begin

V := CreateOleObject('Word.Basic');

V.Insert('Hello from Delphi');

V.FilePrint;

V.FileSaveAs('C:\Hold.doc');

end;

end.

**تحذير :** يجب أن يكون لديك Microsoft Word 95 أو نسخة أقدم مركبة وتعمل لكي تستخدم التطبيق OleWord1 انظر الـ OleWord2 في هذا الباب إذا كان لديك Word 97 أو نسخة أحدث.

Caution

إن عرض البرنامج بسيط ، لذا فهو غير موضح هنا . عندما يبدأ البرنامج في التشغيل ، فهو يعرض زراً كبيراً . اضغط هذا الزر لإنشاء ، طباعة ، وحفظ وثيقة الـ Word . إن OnClick للبرنامج يوضح كيفية استخدام الـ CreateOleObject . كما ذكرت ، يجب أن تضيف تعريف الـ uses لتتمكن من الوصول الى هذه الـ function - يمكنك أن تضيف اسم الـ unit الى تعريف الـ uses الرئيسي للـ module في أعلاها ، أو يمكنك إضافة تعريف الـ Uses كما هو موضح في القائمة بعد كلمة الـ implementation الرئيسية للـ module مباشرة .

لحمل نتائج الـ CreateOleObject، قم بتعريف متغير من نوع الـ Variant مثل هذا:

```
var
  V: Variant;
```

إن الـ Variant هو نوع بيانات خاص وهو، فى الأصل، يمكن أن يكون أى نوع object فعلى. على سبيل المثال، يمكن أن يحمل الـ Variant قيمة عدد صحيح، رقم دى Point، String. ويمكن أيضاً أن يحمل OLE objects - أو لنكون أكثر دقة، IDispatch object مثل الذى تم إدخاله بواسطة الـ CreateOleObject function. إن الـ Variant objects تؤدي التحويل تلقائياً بحيث إذا حمل واحداً عدد صحيح وقمت أنت بتعيينه الى String، يتم تحويل قيمة العدد الصحيح بصورة تلقائية الى نص.

تحذير: قد تبدو الـ Variant وكأنها مسحورة، ومن الطبيعى أن يغريك أن تستخدمها بشكل مكثف. ولكن، أنها كبيرة وبطيئة - فكل Variant له طول ١٦ Byte على الأقل، بغض النظر عن نوعه - والـ Code المرتبطة بعمليات الـ Variant تستغرق وقتاً طويلاً للتشغيل عن الـ Code التى تؤدي على أنواع بيانات أصلية. لا تستخدم الـ Variant إلا إذا كان يجب عليك هذا - لحمل الـ OLE objects، مثلاً.

بعد تعريف Variant، استدع الـ CreateOleObject وعين النتيجة الى الـ Variant. يمكنك عندئذ استدعاء methods والوصول الى خصائص فى الـ OLE object. على سبيل المثال، إن السطرين التاليين ينشئان Word Basic object ثم يستدعيان الـ Insert method لتلك اللغة لإدخال بعض النص فى وثيقة Word:

```
V := CreateOleObject('Word.Basic');
V.Insert('Hello from Delphi');
```

إنك قد تتعصب، كيف عرف الـ Compiler أن الـ Word.Basic لها method يدعى Insert؟ والإجابة هى: إنه لا يعرف. عندئذ استخدم الـ Variants، يقوم Delphi Compiler بإرخاء كل قواعد فحص الـ Syntax الخاصة به ويمكنك من كتابة أى شئ بالإشارة الى الـ Variant. يتم Compile هذه العبارة بشكل سليم:

```
V.NoSuchMethod('Any parameter', 123);
```

## الباب السادس عشر : التطوير مع الـ Clipboard، الـ DDE، والـ OLE

بالرغم من أن الـ Word.Basic ليس لديها هذا الذى يسمى NoSuchMethod. ولكن، عندما تقوم بتشغيل البرنامج، فانك تتلقى رسالة خطأ بهذا المعنى عندما يحاول البرنامج أن يستدعى هذا الـ method فى الـ OLE Word.

وهذه العملية هى مثال على الـ late binding-meaning، أنه بدلاً من حل استدعاءات إلى الـ methods و references أخرى أثناء الـ compilation، فإن الحل للـ Variants يحدث فى وقت التشغيل. عندما يستدعى البرنامج الـ CreateOleObject، إذا كان ناجحاً، فإن هذا الـ object يوفر، عبر واجهة التطبيق الخاصة به، قائمة الـ methods والخصائص المتاحة. وهذه هى الوسيلة التى يمكن بها ان تعمل عبارة مثل الـ V.Insert.

وتوضح العبارتان الاخرتان فى الـ OnClick للبرنامج إستدعاءين آخرين للـ Word Basic. وهذه العبارات:

```
V.FilePrint;
```

```
V.FileSaveAs('C:\Hold.doc');
```

تستدعى الـ FilePrint لطباعة نص الوثيقة والـ FileSaveAs لحفظه فى الملف المشار إليه. إن كل هذه الأعمال تقع فى الـ OLE داخل الـ Microsoft Word.

**ملحوظة:** لا يمكنك تشغيل أى تطبيق قديم بإنشاء OLE. يجب أن يكون التطبيق OLE server مثل الـ Word أو الـ Excel الذين صمما ليستجيبا إلى مطالب تطبيقات عميل الـ OLE مثل برنامج العرض فى هذا الباب.



### OLE object العينة لـ Word 97 وما أحدث:

**على القرص المدمج:** إذا كان لديك Word 97 أو نسخة أحدث، قم بتحميل ملف مشروع الـ Source\OleWord2 إلى Delphi. يبدأ الـ Word - أحب أن يكون عاملاً حتى يعمل هذا البرنامج بشكل سليم. كذلك، قم بتشغيل طابعتك - إذا لم يكن لديك واحدة، فانك تتلقى رسالة خطأ من نظام التشغيل. (إن هذه غير ضارة) ولكنك يمكنك أن تحذف عبارة الـ V.PrintOut لتجنبها.





اضغط F9 لتشغيل البرنامج، ثم اضغط زر إنشاء وثيقة Word، أدخل النص "Hello from Delphi"، اطبع الصفحة، واحفظ الوثيقة على أنها C:\Hold.doc. توضيح القائمة (١٠-١٦) الـ source code للبرنامج.

**تحذير:** يجب أن يكون لديك Microsoft Word 97 أو نسخة أحدث مركبة وتعمل حتى تستخدم التطبيق OleWord2 أنظر OleWord1 فى هذا الباب إذا كان لديك Word 95 أو نسخة أقدم.



القائمة (١٠-١٦) OleWord2\Main.pas

unit Main;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics,  
Controls,  
Forms, Dialogs, StdCtrls;

type

TMainForm = class(TForm)  
  Button1: TButton;  
  procedure Button1Click(Sender: TObject);  
  private  
  { Private declarations }  
  public  
  { Public declarations }  
end;

var

MainForm: TMainForm;

implementation

uses ComObj; // Declares the CreateOleObject function

{\$R \*.DFM}

{ This procedure works with the newer Word 97,  
  which uses Visual Basic in place of the standard

## الباب السادس عشر : التطوير مع الـ Clipboard، الـ DDE، والـ OLE

```
Word Basic as its controlling language. }  
procedure TMainForm.Button1Click(Sender: TObject);  
var  
  V, X: Variant;  
  S: String;  
begin  
  V := CreateOleObject('Word.Document');  
  X := V.Range(0, 0);  
  X.InsertBefore('Hello from Delphi');  
  X.Font.Name := 'Arial';  
  X.Font.Size := 18;  
  X.InsertParagraphAfter;  
  V.Printout;  
  V.SaveAs('C:\Hold.doc');  
end;  
  
end.
```

إن هذه الطبعة من برنامج OLE تشبه السابقة، ولكنها توضح كيفية استدعاء methods والوصول إلى قيم خصائص في الـ Visual Basic، التي تحل محل الـ Word Basic في منتجات برمجيات الـ Microsoft. لسوء الحظ، إن أغلب الأوامر مختلفة. لذا فلنكني تدعم كلا النسختين من الـ Word، فإنك تحتاج إلى تطبيقات منفصلة أو خياراً لاختيار الـ products الصحيح.

إن عملية إنشاء الـ Word 97 OLE object تشبه العملية الخاصة بالـ Word 95، ولكن بدلاً من الـ Word.Basic، فإن الـ CreateOleObject تطلب الآن object من نوع الـ Word.Document. كما سبق، قم لتعريف Variant (وهو V في الـ procedure، ثم استدع الـ CreateOleObject كهذا.

```
V := CreateOleObject('Word.Document');
```

إن تمرير الـ Word.Document كـ string argument كوسيط يضمن أحدث نسخة تم تركيبها من الـ Word يتم استخدامها لإنشاء الـ OLE object. يمكنك عمل مرجع لنسخة برنامج خاص بربط الرقم الكبير للنسخة الخاصة به. على سبيل المثال، إن هذا أيضاً يعمل:

```
V := CreateOleObject('Word.Document.8');
```

بعد انشاء ال OLE object ، يمكن للبرنامج ان يقوم باستدعاء methods . ولكن تذكر ، إن ال methods والخصائص تختلف فى ال Word Basic عنها فى ال Visual Basic . والأسوأ من ذلك ، إن الخطوات المطلوبة لأداء مختلف العمليات اصبحت الآن مختلفة أيضاً . على سبيل المثال ، لإدخال بعض النص الآن فهذا يتطلب على الأقل ثلاثة أوامر مثل :

```
X := V.Range(0, 0);
X.InsertBefore('Hello from Delphi');
...
X.InsertParagraphAfter;
```

إن الاستدعاء للـ Range يمنع استبدال أى نص مختار ، والذي لا يعد ضرورياً فى هذه الحالة ، ولكنه مرشحاً فى دليل برمجة ال Visual Basic . والزر الـ Visual Basic يوضح أين يمكن ان تذهب الأوامر الأخرى لتغيير جوانب من النص الذى تم ادخاله ، والذي تم اختياره طبقاً للنظام الافتراضى . إن البرنامج يستخدم هذا المكان لاختيار اسم font وحجم بالعبارات :

```
X.Font.Name := 'Arial';
X.Font.Size := 18;
```

إن استدعاء ال InsertParagraphAfter يبدأ فقرة جديدة وهو مساو إلى ضغط مفتاح ال Enter عند كتابة وثيقة . لطباعة النص الذى تم إدخاله وحفظه فى ملف ، يقوم البرنامج بتنفيذ إثنين من أوامر ال Visual Basic الأخرى :

```
V.Printout;
V.SaveAs('C:\Hold.doc');
```

**فكرة :** لا يوجد متسع هنا لنخوض فى برمجة ال Visual Basic Tip بالتفصيل . لمزيد من المعلومات ، أنظر أفكار برمجة وتراكيب ال Visual Basic فى online help للـ Word 97 .

### استخدام ال OLEObject :

إن التطبيقين السابقين قد أوضحا كيفية إستدعاء ال CreateOleObject . وهذه الـ function تنشئ object واحداً ، والذي لم يتم بدءه ، باستخدام اسم ال

## الباب السادس عشر : التطوير مع ال Clipboard ، ال DDE ، وال OLE

class والذي يتم تمريره على أنه string parameter . واسم ال class هذا يجب أن يكون مسجلاً في ال Windows registry . احفظ نتيجة ال function في Variant . على سبيل المثال ، استدع ال function مثل هذا لإنشاء وثيقة Word .

```
V := CreateOleObject('Word.Document');
```

إن ال Word.Document string يعد بالفعل تمثيلاً للـ Class ID (CLSID) ، والذي يسمى في بعض الأحيان بـ (goosey ID) GUID . وهذا ال ID هو hexadecimal key والذي هو مضمون من الناحية الاحصائية أن يكون فريداً . (إذا تداخل أى اثنين من ال GUIDs ، تكون النتيجة بلا شك فوضى كبيرة) . باستخدام ال Regedit.exe utility في ال C:\Windows ، يمكنك أن تنظر في ال CLSID للبحث عن البرامج المركبة . عندما فعلت هذا ، لقد بحثت في ال HKEY\_CLASSES\_ROOT (المدخل الجذرى للـ classes المسجلة ومفاتيحها) عن ال Word.Document ، والذي عرض ال CLSID التالى :

```
"{00020906-0000-0000-C000-000000000046}"
```

هذا هو نفس ال CLSID للـ Word.Document.8 . وهذا ما يفسر أن أى من التصميمين يعمل لغرض إنشاء ال OLE . فى المستقبل ، إذا قمت بتركيب نسخة أحدث من ال Word ، فأتوقع أن ال CLSID الخاص بالـ Word.Document سيتم تحديثه فى ال Windows registry .

إن ال CreateOleObject function يتم تعريفها فى ملف ال ComObj.pas التابع لـ Delphi كما يلى :

```
function CreateOleObject(const ClassName: string): IDispatch;
```

وتقوم ال function بإدخال object من نوع ال IDispatch ، والتى تم تعريفها كواجهة تطبيق مشتقة من ال IUnknown :

```
IDispatch = interface(IUnknown)
```

```
['{00020400-0000-0000-C000-000000000046}']
```

```
function GetTypeInfoCount(out Count: Integer): HRESULT;  
stdcall;
```

```
function GetTypeInfo(Index, LocaleID: Integer;
```

```

out TypeInfo): HRESULT; stdcall;
function GetIDsOfNames(const IID: TGUID; Names:
Pointer;
NameCount, LocaleID: Integer; DispIDs: Pointer): HRESULT;
stdcall;
function Invoke(DispID: Integer; const IID: TGUID;
LocaleID: Integer; Flags: Word; var Params;
VarResult, ExcepInfo, ArgErr: Pointer): HRESULT; stdcall;
end;

```

إن هذا الـ object يستخدم بعض التراكيب الغريبة والتي لن تجدها في كثير من الأماكن في Delphi. إنك لست في حاجة إلى أن تفهم كل التفاصيل الموجودة هنا: فقط كن على حذر عندما تستدعي الـ CreateOleObject، فأنك تحصل مرة أخرى على object من نوع الـ IDispatch. وهذا الـ object يشير إلى الـ OLE، الذى يكمن فى التطبيق المضيف له. إن كل الاستدعاءات لهذا الـ object، عبر واجهة تطبيق OLE object، يتم فعلها باستدعاء function الاستدعاء فى الـ IDispatch.

ليس من الضرورى حقاً أن تفهم كل هذه الموضوعات المعقدة المتعلقة بالسجل وقيم الـ CLSID. ولكن، قبل الغوص فى بحور برمجة الـ OLE، يجب أن تفهم أن الـ OLE objects توجد فى برامجها المسجلة وأن object تطبيقك هو بالفعل من نوع الـ IDispatch. ولأنك لا تستطيع فى أن تعرف حتى وقت التشغيل ما هو واجهة التطبيق التى يوفرها تطبيقك الـ OLE، فعليك أن تستخدم Variant لتقوم باستدعاءات OLE object من خلال الـ IDispatch. إذا واجهت مشكلة، استخدم Windows Regedit utility لتتصفح أسماء الـ objects المتاحة التى يمكن لتطبيقك أن ينشئها، ولكى تزيل المشكلات. إذا لم يكن الـ object المسجل له CLSID، فلا يمكنك أن تنشئ له OLE، وحتى لو كان له، فلا يوجد ضمان. أن تمرير اسم الـ object class للـ CreateOleObject يؤدي إلى نتيجة. (هناك مدخل للـ Word.Basic على سجل النظام الخاص بى، ولكن الـ CreateOleObject لا يمكنها أن تنشئ OLE من هذا النوع).

**تحذير:** كن على حذر عند استخدام برنامج ال Regedit . إن التصفح أمر مسموح ، ولكن حتى التغييرات البسيطة Windows registry قد يكون لها نتائج مدمرة . لمزيد من السلامة ، استخدم أمر ال Registry File Export Registry File الخاص بال Regedit لإنشاء نسخ نص احتياطية للـ Registry بأكمله . يمكنك عندئذ استخدام أمر ال Registry Import Registry File لتحميل ال Registry الذى تم حفظه . انظر أيضاً عنوان مساعدة ال Regedit وهو "Restoring the registry" لمعرفة الخطوات التى يمكنك استخدامها لاستعادة سجل تم اتلافه .

تقوم ال CreateOleObject بادخال reference لتعريف واجهة التطبيق والذى يمكن استخدامه للاتصال بال object . ولـ CreateOleObject تكون واجهة التطبيق من نوع ال IDispatch . لإنشاء object COM ليس من واجهة تطبيق ال IDispatch ، استخدم ال CreateComObject .

### كتابة تطبيق حاوية OLE :

إن الفصول السابقة قد أوضحت طريقة واحدة فقط لاستخدام ال OLE . وهناك طريقة أخرى وهى انشاء تطبيق حاوية يتصل بال OLE server . وبهذه التقنية ، يمكنك ربط وتضمين وثائق server فى تطبيقك . على سبيل المثال ، تستطيع حاوية ال OLE أن تنشئ ، تحمل ، تحرر ، وتحفظ وثيقة ال Microsoft Excel . ولأن ال Excel هو تطبيق OLE server مكتمل الصفحات ، يستطيع برنامج Delphi الخاص بك أن يستدعى ال Excel لإنشاء وثائق ، جميعها من داخل نافذة برنامجك .

إن ربط وتضمين وثائق من أنواع مختلفة - وخاصة إذا كانت تلك الأنواع غير معروفة فى وقت التصميم - هى طريقة رائعة لتمكين المستخدمين من أن يقرروا كيف يتمنون أن يخزنوا المعلومات على حاسباتهم الإلكترونية . يمكنك أيضاً أن تستخدم الربط والتضمين كطريقة لتوفير أوامر معقدة فى برامجك بأقل قدر من البرمجة . بدلاً من قضاء نصف اليوم فى انشاء محرر نص للمستخدمين لادخال وتحرير وثائق ، يمكنك ببساطة أن توفر حاوية OLE وتمكن مستخدمى برنامجك من إنشاء وثائق نص باستخدام المصدر المفضل لديهم - سواء كان هذا Microsoft Word أو WordPerfect .

فى هذا الفصل ، سوف أوضح كيفية انشاء تطبيق حاوية OLE عامة باستخدام الـ OleContainer من الـ System palette . والبرنامج يوضح كيفية تنفيذ الـ in-place editing ، والذي بواسطته يتم استبدال قوائم البرنامج و toolbars الخاص به الـ OLE server . عندما تفتح الـ Excel ، تظهر الـ toolbars وقوائم الـ Microsoft Excel داخل تطبيق الـ Delphi . وبنفس الطريقة ، تفتح وثيقة Word ، تحمل أدوات وقوائم الـ Microsoft Word محل واجهة التطبيق لتطبيق الـ Delphi . ويوضح البرنامج أيضاً كيفية تمكين المستخدمين من تضمين وثائق كأيقونات فى التطبيق ، بحيث إذا تم فتحها فهى تحضر تطبيقات الـ Server الخاصة بها فى نوافذ منفصلة .

لتوفير تحرير الـ OLE in-place editing ، يجب أن يحتوى تطبيق الـ Delphi على :

\* OleContainer object واحد لكل الـ OLE ibject فى الحاوية .

\* MainMenu object .

\* toolbar Panel object (إختياري) .

\* statusbar Panel object (إختياري) .

بهذه البنود ، يستطيع الـ Server أن يدخل القوائم الخاصة به الـ toolbar والـ SpeedButtons ، والـ Status Bar فى الحاوية . على سبيل المثال ، عند تحرير الـ Microsooft Word متضمن أو مربوط فى حاوية ، فإن قوائم الـ Word تزيد على قوائم الحاوية . ويرى المستخدمون أوامر البرنامج المعالج للكلمات والمألوف لهم ، ولكنهم لا يزالون يقومون بتشغيل الحاوية - فلا يجب عليهم التحول الى أو تشغيل الـ Server باستخدام الـ toolbar Panel خاص بالحاوية ، فيجب أن تكون خاصية الـ Align له مساوية لـ alBottom أو alLeft أو alRight أو alTop ، ويجب أن تكون الـ Locked مساوية لـ False . كذلك ، يجب أن تستخدم الحاوية جهاز الـ MDI للتحكم فى الـ Client area بالنافذة الرئيسية ولتوفير الـ Child window لتحرير الوثائق . من الممكن أن تستخدم تطبيق احادى النافذة مثل عميل حاوية الـ OLE ، ولكن واجهة تطبيق الـ MDI أكثر ملائمة لأنه يمكن المستخدمين من إنشاء وثائق الـ Child window من أى نوع الـ Server .

## الباب السادس عشر : التطوير مع ال Clipboard ، ال DDE ، وال OLE

لإنشاء حاوية OLE ، أضف instance من ال OleContainer على form . إن ال object الناتج يشبه ال Panel ذا سطح مفرغ . وفيه يستطيع المستخدم أن يدخل OLE ، والمعرض كأيقونة أو في Form كاملة .  
قم بتجربة هذه الخطوات لتكتب تطبيق حاوية OLE هيكلي يوضح أسس العمل بال OLE object :

١ - إبدأ تطبيقاً جديداً وأضف OleContainer من لوحة ال System VCL في نافذة Form1 .

٢ - أضف MainMenu من ال Standard palette . اضغط مرتين أيقونة القائمة ، وادخل قائمتين : File و Edit . حدد خاصية ال GroupIndex لقائمة ال Edit بواحد . في قائمة ال File ، ادخل أمر Exit . في ال Edit ، ادخل أمر Insert . Object....

٣ - اترك ال Menu Designer واختر File|Exit من قائمة ال form . ادخل Close بين ال begin وال end في ال event handler لأمر القائمة . اختر Edit|Insert Object... من قائمة ال form ، وادخل البرمجة من القائمة (١٦-١١) . إن النص مأخوذ من ملف ال Source\OleCont\Main.pas عيى القرص المدمج .

القائمة (١٦-١١) هذا الإجراء يوضح الخطوات المطلوبة  
لإدخال OLE في OleContainer component .

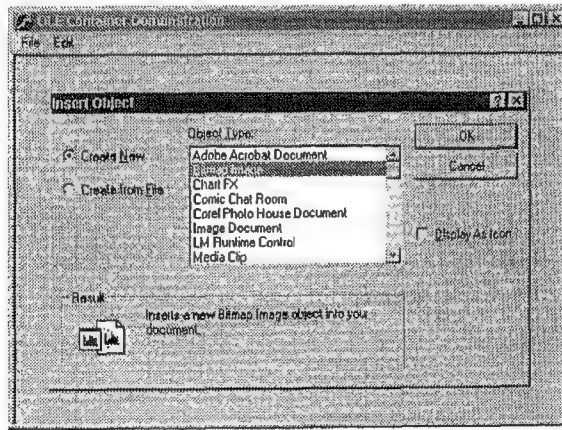
```
procedure TMainForm.InsertObject1Click(Sender: TObject);
begin
    with OleContainer1 do
    begin
        if InsertObjectDialog then
            DoVerb(PrimaryVerb);
    end;
end;
```

إن ال event handler لأمر ال Edit|Insert Object.. يوضح الطريقة الأساسية لاستخدام OleContainer . استدع ال InsertObjectDialog



method الخاص بال object ، والذي يعرض ال dialog الموضح فى شكل (٦-١٦) . يستطيع المستخدم أن يختار نوع ال OLE لإنشاء - Bitmap مثلاً - وأيضاً يختارها إذا كان سيدخل صورة كاملة لل object أو يعرضه كأيقونة . يمكن للمستخدمين أيضاً أن يختاروا أيقونة لإنتقاء الملف . وكل هذه الأعمال تقع فى مستوى نظام التشغيل - إن ال code تقوم باستدعاء ال InsertObjectDialog لتبدأ العجلة فى الدوران .

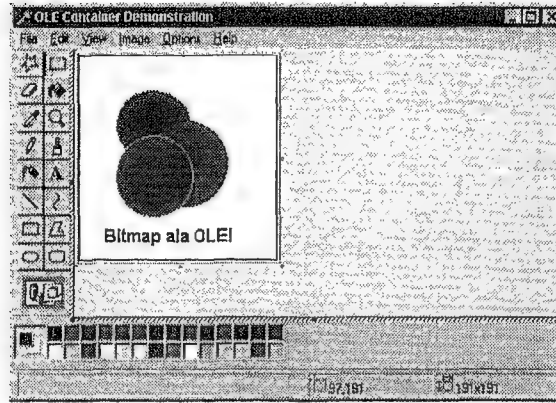
إذا رجعت هذه ال function بـ True ، يكون المستخدم قد اختار object لإنشاءه . لإنهاء العمل وإدخال ال OLE object فى OleContainer فى تطبيق Delphi ، استدع DoVerb وقم بتمرير ال argument PrimaryVerb إن ال DoVerb method يؤدي عملاً لل OLE object إن ال PrimaryVerb هو العمل الافتراضى حسب النظام والوارد لهذا النوع من ال objects - عادة مايقوم هذا الفعل بإنشاء وبدء object جديد .



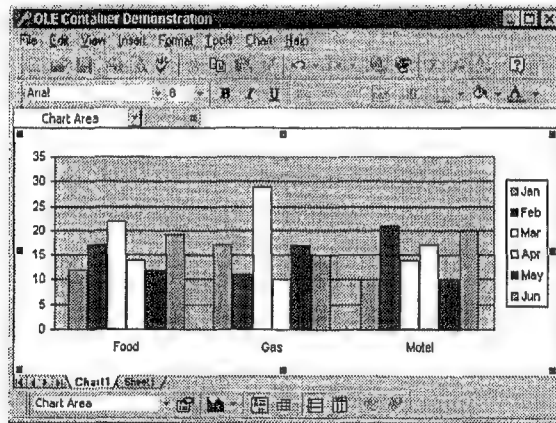
شكل (٦-١٦) ال Insert Object Dialog معروض بواسطة OleContainer

يوضح الشكل (٧-١٦) تطبيق ال OleCont بعد ان قمت بإنشاء ملف dialog كاستجابة لل dialog الموضح فى شكل (٦-١٦) . Windows Paint utility - وهى OLE server تحمل محل قوائم تطبيق Delphi وتعرض مع قوائم التطبيق الخاص بها . ويقدر يسير من البرمجة ، يوفر البرنامج امكانيات تحرير ملف جرافيك كاملة للمستخدمين . يوضح الشكل (٨-١٦) نفس البرنامج ، ولكن هذه المرة عارضاً وثيقة Microsoft Excel Chart . ألا تأمل أن تكون كل البرمجة بهذه السهولة؟

## الباب السادس عشر : التطوير مع الـ Clipboard، الـ DDE، والـ OLE



الشكل (١٦-٧) تطبيق الـ OleCont محرراً ملف bitmap باستخدام الـ Windows Paint



الشكل (١٦-٨) نفس تطبيق الـ OleCont ولكن هذه المرة محرراً وثيقة Microsoft Excel Chart

توضح القائمة (١٦-١٢) البرمجة التي كانت مطلوبة في Delphi لإنشاء وثائق OLE متضمنة بالرغم من أن OleContainer يتم استخدامه بنفس الطريقة، إلا أن البرمجة أصبحت مختلفة للغاية وهي موضحة هنا لأغراض المقارنة فقط. لا تستخدم هذه التقنيات في التطبيقات الجديدة. انظر القائمة (١٦-١١) لمعرفة الـ methods الصحيحة التي يجب استخدامها لتضمين الـ OLE objects في تطبيقات حاوية.

### القائمة (١٦-١٢) تضمين ال OLE object

فى OleContainer يتطلب استخدام Code مثل هذه.

```
procedure TForm1.InsertObject1Click(Sender: TObject);
var
  P: Pointer;
begin
  if InsertOLEObjectDlg(Form1, 0, P) then
    begin
      OleContainer1.PInitInfo := P;
      ReleaseOleInitInfo(P);
    end;
end;
```

### دمج قائمة ال OLE:

لتوفير ال OLE 2.0 in-place editing ، يقوم ال MainMenu الخاص بتطبيق الحاوية بتعيين قيم معيارية لخصائص ال GroupIndex لعنصر قائمة عالية المستوى . عندما يقوم المستخدمون بتصوير object ، يحل ال OLE 2.0 Server محل القوائم ذات ال GroupIndex المساوية ، ويدخل قوائم إلى اليسار من القوائم التى لها GroupIndex اعلى . ويوفر المساعد أيضاً أوامر ال Update وال Exit to Container أو الأوامر المشابهة ، عادة فى قائمة ال File ، لحفظ ال objects الجديدة أو المعدلة فى تطبيق الحاوية .

يذكر الجدول (١٦-٢) الأوامر والقوائم العالية المستوى المقترحة لتطبيق حاوية ال OLE . حدد ال GroupIndex الخاصة بعنصر القائمة العالية المستوى بالقيم الموجودة فى الجدول لتمكن من دمج قائمة ال OLE 2.0 server . على سبيل المثال ، حدد ال GroupIndex الخاصة بالقائمة Edit بواحد حتى يستطيع المساعد أن يستبدل هذه القائمة بالأوامر الخاصة بها .

جدول (١٦-٢) قيم خاصية ال GroupIndex وقوائم تطبيق حاوية ال OLE 2.0 المقترحة

القائمة	الأوامر المقترحة	ال GroupIndex
File	New, Open, Exit	0
Edit	Insert object, Paste special, Object (set Enabled to false)	1

## الباب السادس عشر : التطوير مع ال Clipboard ، ال DDE ، وال OLE

Object	Deactivate	2
View	none	3
Window	Cascade, Tile, Arrange icons	4
Help	none	5

**Tip . فكرة :** لا يجب عليك أن تحدد كل قيمة GroupIndex تابعة لعنصر

قائمة . قم بتعيين قيم فقط لبنود القائمة العالية المستوى .

قم بتعيين اسم قائمة ال Object الخاصة بتطبيقك ، بقيمة GroupIndex مساوية لاثنتين ، إلى خاصية ال MenuItem التابعة لل form . فى التطبيق الأحادى الوثيقة ، عين هذه الخاصية لل form الرئيسية . فى تطبيق ال MDI ، عين الخاصية فى ال Child form لبرنامجك . بالقيام بهذه التعيينات ، يقوم ال OleContainer فى ال form بصورة تلقائية بجعل أوامر تحرير ال object المختارة .

استخدم قيم ال GroupIndex بدلاً من تلك الموجودة فى جدول (١٦-٢) لمنع تطبيقات ال OLE 2.0 server من ان تحل محل قوائم التطبيق الخاصة لك . على سبيل المثال ، إذا قمت بتعيين القيمة تسعة لخصائص ال GroupIndex لكل قائمة عالية المستوى ، يتم إضافة بنود قائمة ال server إلى أوامر قائمة التطبيق الخاص بك . خفض إمكانية الخلط (إثنين من قوائم ال Edit ، مثلاً) ، إذا قمت بهذه الحيلة ، فلا يجب أن يستخدم تطبيقك أى من أسماء القائمة المذكورة فى الجدول .

### ال OLE وال clipboard :

يجب عليك ان تضيف امر ال Paste Special لقائمة ال Edit الخاصة بتطبيقك بحيث يستطيع المستخدمون أن يلصقوا ال OLE من ال Windows clipboard . ليس كل تطبيقات ال OLE server تستطيع نسخ ال objects إلى ال clipboard ، ولكن هذا الأمر يمكن المستخدمين من لصق ال objects للتطبيقات التى تستطيع ذلك . إنك تحتاج إلى أمر ال Paste Special ، استدع ال Paste Special Dialog الخاصة بال Ole Container . إنك أيضاً عادة ما تريد أن تتأكد إن كانت الحاوية بها وثيقة حالياً ، وفى هذه الحالة يجب حث المستخدمين على التخلص منها . توضح القائمة ١٦-٣ التخطيط العام لأمر ال PasteSpecial .

القائمة ١٦-٣: نفذ أمر PastelSpecial باستخدام code مثل هذه.

```
procedure TMainForm.PasteSpecial1Click(Sender:
TObject);
begin
  if (OleContainer1.State = osEmpty) or
    (MessageDlg('Delete current OLE object?',
      mtConfirmation, mbOkCancel, 0) = mrOk) then
    if OleContainer1.PasteSpecialDialog then
      begin
        // ... Enable Cut, Copy, and Paste buttons here
      end;
    end;
```

هذا ال code يشبه تلك الموجودة فى برنامج العرض الخاص بـ Delphi، وهو OleSdi الموجود فى دليل Demos\Olectnrs. يتم فحص خاصية ال State أولاً لتحديد ما إذا كان ال OleContainer خالياً. إذا لم يكن خالياً، يقوم dialog رسالة بحث المستخدمين على اذن بحذف ال OLE الحالى. إذا اجاب المستخدم بـ yes (نعم)، يقوم البرنامج باستدعاء PasteSpecialDialog، والذي يتولى باقى المهام المطلوبة. ويوضح الزر البيضوى أين يمكنك أن تدخل code لتشغيل أزرار أو أوامر قائمة متنوعة الآن حيث قام المستخدم بلصق object فى الحاوية. على سبيل المثال، إذا كان لديك CopyButton، يمكنك تنفيذ هذا ال code:

```
CopyButton.Enabled := True;
```

كن على حذر من أن ليس كل ال OLE servers يمكنهم لصق objects، وبدلاً من تشغيل زر ال Paste أوامر قائمة، فمن الأفضل أن تستخدم خاصية ال CanPaste فى الحاوية. وهذه الخاصية تكون True إذا كان ال OLE server يمكن من اللصق. قم بتشغيل ال PasteButton باستخدام code مثل هذه:

```
PasteButton.Enabled := OleContainer1.CanPaste;
```

## افكار للمستخدم الخبير

\* في البرامج التي تلتصق objects كبيرة مثل ال Bitmaps إلى ال clipboard ، فقد تريد أن تمنح المستخدمين فرصة مسح ال clipboard قبل أن ينتهي البرنامج . وهذا يساعد على توفير الذاكرة . على سبيل المثال ، أدخل استفساراً مثل "Clear large bitmap on clipboard?" في ال OnClose الخاص بال form الرئيسية .

\* بالرغم من أن ال DDE يمكن أن ينقل بيانات النص فقط ، فإن ذلك النص قد يمثل قيم عددية ، أسماء ملفات ، أو هياكل للبايت ممثلة في hexadecimal strings . وقد يتطلب الأمر بعض البرمجة لاستخدام ال DDE لنقل ال objects الثنائية ، ولكن إذا عرفت كيف تمثل بياناتك في ال text object ، فيستطيع ال DDE حمل المعلومات .

\* عند الاتصال بال servers الذين لا يستخدمون أسماء الملفات الخاصة بهم مثل ال Service ، في ال DdeClientConv لتطبيق العميل ، حدد ال DdeService بالمعرف الذي تم الحصول عليه من وثائق ال server . حدد ال ServiceApplication باسم ملف ال server . يستطيع العميل عندئذ أن يبدأ ال server إذا كان ال ConnectMode ، والموجود هو الآخر في ال DdeClientConv محدد بـ ddeAutomatic .

\* لإضافة scroll bars للـ OLE objec ، أضف نموذج من ال OleContainer في ScrollBox . لكي تجعل scroll bars مرئية ، حدد الخصائص الفرعية Range للـ HorzScrollBar و VertScrollBar بقيم ScrollBox على التوال .

\* عند برمجة تطبيقات حاوية ال OLE الأحادية النافذة ، فقد يكون عرض برنامجك أفضل إذا وضعت ال OleContainer في Panel . حدد خاصية ال Align للـ Panel بـ alClient ، ثم قم بإسقاط OleContainer في ال Panel . إنك لا تحتاج أن تقوم بهذا إذا كان تطبيقك يستخدم تصميم ال MDI المتعدد النوافذ لأنه ، في هذه الحالة ، كل Child window يملك OleContainer منفصل .

## المشروعات التي يمكنك تجربتها

١٦-١: اكتب متصفح Clipboard الخاص بك الذى يوفر forms معيارية وكذلك TComponent و TPicture.

١٦-٢: اكتب تطبيق DDE server الذى ينشئ خطاباً فى الـ Microsoft Word أو أى برنامج معالج كلمات آخر. قد يقدم برنامجك أنواع مختلفة عديدة من الـ forms.

١٦-٣: اكتب تطبيق عميل DDE الذى يطبع بطاقات أو أسماء أو عناوين ثم إدخالها فى الـ Microsoft Excel أو أى تطبيق ورقة حسابات آخر.

١٦-٤: متقدم: اكتب تطبيق حاوية OLE قراءة فقط والذى يربط اثنين أو أكثر من الوثائق من تطبيقات الـ OLE server على نظامك على سبيل المثال وثيقة معالجة كلمات وورقة عمل، مثلاً.

## ملخص:

\* إن الـ Windows clipboard، تعتبر اسهل طريقة للتشارك فى البيانات فيما بين التطبيقات. تتطلب الـ clipboard التعاون التطوعى لإثنين من التطبيقات (أو اثنين من العمليات فى نفس التطبيق).

\* إن الـ Dynamic Data Exchange، أو الـ DDE، تقدم طريقة أكثر تعقيداً للتشارك فى بيانات نص. لاستخدام الـ DDE، يقوم تطبيق عميل وتطبيق server باقامة محادثة تنتقل عبرها البيانات. تنتقل البيانات عادة من الـ إلى العميل، ولكن التطبيقات تستطيع أن تعكس أدوارها وتكرر البيانات فى الاتجاه الآخر.

\* يقوم الـ Object Linking and Embedding، أو الـ OLE، بتحويل التركيز من التطبيق إلى الوثيقة. وكذلك، مع الـ OLE، يستطيع المستخدمون إنشاء أنواع ووثائق جديدة بطرق غير متوقعة من قبل مصممي التطبيقات.

## الباب السادس عشر : التطوير مع الـ Clipboard، الـ DDE، والـ OLE

\* إن تطبيق حاوية الـ OLE يربط ويضمن الـ objects. يوفر تطبيق الـ OLE server التحرير وأوامر أخرى لأنواع خاصة من الـ objects. استخدم الـ OleContainer التابع لـ Delphi لتنشئ تطبيقات حاوية الـ OLE.

إن كل حاسوب يحتاج إلى نظام قاعدة بيانات جيدة، وكما ستكتشف في الباب التالي، فإن Delphi يوفر أدوات تطوير قاعدة البيانات التي تعتبر نافعة ومفيدة.





# الباب السابع عشر

## تطوير تطبيقات قاعدة البيانات

### محتويات هذا الكتاب:

- Components.
- تطوير قاعدة البيانات.
- Database components.
- Structured query language.
- Master-detail databases.
- استخدام Data Modules.
- موضوعات برمجة قاعدة البيانات.

بالرغم من أن Delphi يعد أحد أعمدة برمجة الـ Windows ، إلا أنه ، وكما يوضح هذا الباب ، يعتبر مطوراً لقاعدة البيانات أيضاً . مع Delphi ، يمكنك إنشاء ، تحرير ، وكتابة لكل أنواع قواعد بيانات الحاسبات المكتبية تقريباً مثل نظم الـ Paradox ، dBASE ، ODBC مثل الـ Microsoft Access . يمكنك أيضاً تطوير تطبيقات الـ server والعميل المعقدة . مع Delphi ، إنك لن تحتاج أبداً إلى نظام تحكم آخر في قاعدة البيانات .

في هذا الباب ، سوف أوضح كيفية البدء مع components قاعدة البيانات الخاصة بـ Delphi ، وكيف يمكنك استخدامها لإنشاء واستخدام قواعد البيانات في forms متنوعة . بعد أن تتمكن من بعض الأسس ، سوف تتعرف على كيفية أداء أبحاث مع الـ SQL وكيفية إنشاء تطبيقات قواعد بيانات علاقية تعتمد على نمط master-detail table model .

**ملحوظة:** إن كل نسخ Delphi تتضمن الـ Borland Database Engine ، أو (BDE) ، الذي يوفر مجموعة كاملة من أدوات البرمجة لكثير من نظم قواعد البيانات الحاسبات المكتبية الشهيرة مثل الـ dBase والـ Paradox. تتضمن طبعة الـ Client-Server التابعة لـ Delphi الـ BDE ، وهي تمكن من الوصول إلى server قاعدة البيانات مثل الـ Oracle ، Sybase ، Microsoft SQL Server ، Informix. وتأتي طبعة الـ Client-Server من Delphi أيضاً بـ components لتطويل تطبيقات الـ server والعميل . يمكنك استخدام أى طبعة من Delphi مع هذا الباب .

## Components:

إن Components قاعدة بيانات Delphi هي :

● **BatchMove** - يؤدي العمليات الجماعية على السجلات والجداول ، مثل ازدواج مجموعة بيانات ، إلحاق سجلات من إحدى مجموعات البيانات إلى أخرى ، وتحديث أو حذف سجلات تتماشى مع argument معينة . الـ Palette : Data Access

● **Database** - يوفر خدمات قاعدة بيانات إضافية مثل server logins والأسماء المحلية . يقوم Delphi بإنشاء Database objects بصورة تلقائية على حسب الحاجة ، ولكن يمكنك انشاءها بوفرة إذا لزم الأمر .  
Data Access : Palette

● **DataSource** - يربط components مجموعة البيانات مثل الـ Table والـ Query بالـ data-aware components مثل الـ DBEdit والـ DBMemo . إن كل تطبيق قاعدة بيانات يحتاج على الأقل واحداً من الـ DataSource objects .  
Data Access : Palette

● **DBChart** - وهو component مكتمل الصفات لإنشاء Charts من معلومات قاعدة البيانات . يوضح الباب التالي كيفية استخدام الـ DBChart .  
Data Controls : Palette

• **DBCheckBox** - data-aware CheckBox . Palette: Data Controls .

• **DBComboBox** - data-aware ComboBox component . Palette: data Controls .

• **DBCtrlGrid** - مجموعة من scrollablePanels كل منها تمثل سجل قاعدة بيانات واحد . قد تحتوى كل Panels على objects واحد أو أكثر من ال data-aware control objects . يستخدم هذا ال component مع ال DBNavigator object لتصفح غير تقليدى للبيانات . انظر أيضاً ال DBGrid .  
Palette: Data Controls

• **DBEdit** - component لإدخال نص أحادى السطر data-aware Edit .  
ال Palette: Data Controls .

• **DBGrid** - data-aware ، أعمدة وصفوف تعرض سجلات قاعدة بيانات ، سجل واحد فى كل صف . يستخدم مع ال DBNavigator لتصفح البيانات فى جدول تقليدى ال Palette: Data Controls .

• **DBImage** - data-aware graphical Image يستخدم لعرض ال Blobs (اشكال ثنائية كبيرة) تحتوى على bitmap . Palette: Data Controls .

• **DBListBox** - data-aware ListBox . Palette: Data Controls .

• **DBLookupComboBox** - data-aware ComboBox له القدرة على البحث فى جدول مراجعة . إن النسخ الأولى من Delphi تسمى هذا بـ DBLookupCombo . Palette: Data Controls .

• **DBLookupListBox** - data-aware ListBox ذا قدرة على بحث جدول مراجعة . إن النسخ الأولى من Delphi تطلق على هذا ال component اسم DBLookupList . Palette: Data Controls .

• **DBMemo** - data-aware Memo مدخل نص متعدد الأسطر . Palette: Data Controls .

• **DBNavigator** - أداة تحرير وتصفح معقدة لقاعدة البيانات . ويعتبر هذا بالنسبة لبرمجة قاعدة البيانات مثل التحكم عن بعد بالنسبة لجهاز تسجيل الفيديو .

يضغط المستخدمون أزرار ال DBNavigator للتنقل عبر سجلات قاعدة البيانات ، إدخال سجلات جديدة ، حذف سجلات ، وأداء عمليات أخرى . Palette: Data Controls .

● **DBRadioGroup** - component RadioGroup data-aware .  
Palette: Data Controls

● **DBRichEdit** - مثل ال DBMemo control ، يستطيع ال DBRichEdit أن يعرض بيانات نص مخزن في rich-text format (RTF) . Palette: Data Controls .

● **DBText** - هي data-aware لعرض معلومات قاعدة البيانات التي لا تريد أن يحررها المستخدمون . (ملحوظة : استخدم ال Label المعيارى ، وليس ال DBText ، لوضع label حقول الإدخال على شاشات إدخال البيانات) .  
Palette: Data Controls

● **Query** - يحدد عبارات SQL لل BDE أو لل SQL server . Palette: Data Access

● **Session** - تقع كل اتصالات قاعدة البيانات في موضوعية ال Session ، وهو الذى يتحكم فى هذه الاتصالات . يقوم Delphi بصوره تلقائية بإنشاء Session عام لكل تطبيقات قاعدة البيانات ، ولكن ، يمكنك إضافة Session components لتطبيق ما لتوفر دورات متعددة ، والتي قد تفعلها مثلاً للوصول إلى جداول فى مواضيع مختلفة من الشبكة . Palette: Data Access

● **StoredProc** - يمكن التطبيقات من تنفيذ ال procedures المخزنة على قاعدة بيانات . إذا لم تكن تطور تطبيقات قاعدة بيانات ال server والعميل ، فإنك لن تحتاج غالباً لاستخدام هذا ال component . تتضمن ال procedures المخزنة أوامر للحصول على معلومات حول العمليات المغلقة وأرقام ال IDs لمستخدمى قاعدة البيانات . ولكن ، التطبيق الدقيق المتاح يعتمد على ال server . Palette: Data Access

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

● **Table** - يمنح التطبيقات وصولاً إلى قواعد البيانات من خلال الـ BDE . يرتبط هذا الـ component عادة بالـ DataSource ، الذي يصل الـ Table بالـ data-aware . إن أغلب تطبيقات قاعدة البيانات لها على الأقل Table object واحد . Palette: Data Access .

● **UpdateSQL** - يتوفر هذا للمطورين الذين يحتاجون إلى أداء تعديلات على مجموعة بيانات للقراءة فقط التي يتم إدخالها بواسطة SQL server . باستخدام الـ UpdateSQL يمكن أداء أوامر الـ UPDATE ، INSERT ، DELETE بالرغم من أن مجموعة البيانات تحمل صفة للقراءة فقط (يمكن أن يحدث هذا ، مثلاً ، عندما يقوم التطبيق بتجميع tables متعددة ، بالرغم من أن الـ tables ذاتها ليست للقراءة فقط) إنك تحتاج هذا الـ component فقط إذا كنت واجهت هذا النوع من المشكلات ، وإن لم تكن ، استخدم الـ Query . Palette: Data Access .

**ملحوظة:** الـ Data-aware controls تستطيع استخدام معلومات من

قاعدة البيانات . على سبيل المثال ، إن الـ DBListBox يشبه الـ ListBox العادي ، ولكنه يستطيع أن يحصل على معلوماته من الـ

Table عبر الـ DataSource .

إن النسخ الأولى من Delphi توفر الـ Report في الـ Data Access palette . وهذا الـ component ، والذي كان يوفر الوصول إلى الـ ReportSmith Report التابعة لـ Borland ، والذي لم يعد متوفراً . لإنشاء تعريفات لقاعدة بيانات ، استخدم الـ components الموجودة على الـ QReport palette (Quick Report) . سيتم شرح الـ QReport (والـ TDBChart) في الباب القادم .

## تطوير قاعدة البيانات:

أن components قاعدة البيانات تجعل الوصول إلى قواعد البيانات أمراً معيارياً وذلك بـ forms متعددة . هذا يعني أن تطبيقاتك يمكنها الوصول إلى بيانات في ملفات dBASE ، والـ Paradox Tables ، والـ Microsoft Access ونظم الـ

Open Database Connectivity (ODBC) الأخرى، أو إذا كان لديك طبعة ال Client-Server، فمن خلال ال SQL server.

وافضل من ذلك كله، أنه باستطاعتك استخدام كل ال Delphi components الأخرى، لتقنيات واجهة التطبيق، وبرمجة ال Object Pascal فى تطبيقات قاعدة البيانات الخاصة بك.

### استخدام ال Database Form Wizard:

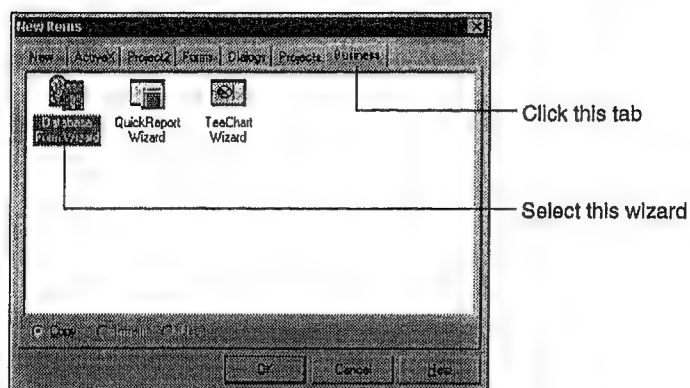
يمكنك تطوير تطبيقات قاعدة البيانات باستخدام أدوات وتقنيات موصحة من خلال هذا الكتاب. يختلف تطبيق قاعدة البيانات عن برامج ال Windows فقط فى قدرته على قراءة وكتابة معلومات فى Tables قاعدة البيانات. ومن خلال واجهة تطبيق المستخدم التابع للبرنامج، فإنك تطور form التطبيق كما تفعل مع أى نافذة أخرى. ولكن، form قاعدة البيانات النموذجية قد تتطلب edit controls، labels، و grids عديدة بالإضافة إلى ال components التى توفر الروابط الضرورية لـ Tables قاعدة البيانات. قد يكون من العسير أن تبرمج كل هذه ال objects على حدى، فيمكنك بدلاً من ذلك أن تستخدم طبعة ال Database Form Wizard. (ينفذ من خلال أمر ال Database Form Wizard التابع لـ Delphi فى قائمة ال Help فى النسخ الأولى) لتستخدم هذه ال Wizard الجديدة، إبدأ تطبيقاً جديداً بأمر ال File | New...، اضغط ال Business Page tab، واختر ال Database Form Wizard.

يعتبر ال Database Form Wizard أداة تفاعلية لإقامة forms قاعدة البيانات. باختصار، إنك تجيب استفسارات متنوعة وتختار خيارات يعرضها ال Wizard. عندما تنتهى من ذلك ال Wizard ينشئ form جديدة تماماً كاملة بكل components قاعدة البيانات فى أماكنها الصحيحة. يمكنك تحريك هذه ال components وعمل تعديلات أخرى للـ form النهائية، ولكن فى كثير من الحالات، تتطلب النتائج النهائية تعديلاً طفيفاً لإنشاء تطبيقاً تاماً مكتملاً.

اتبع هذه الخطوات لإنشاء form قاعدة بيانات لـ Tables فى قاعدة بيانات متوفرة مع ال Delphi (أو يمكنك استخدام Table قاعدة بيانات آخر إذا كان لديك واحداً):

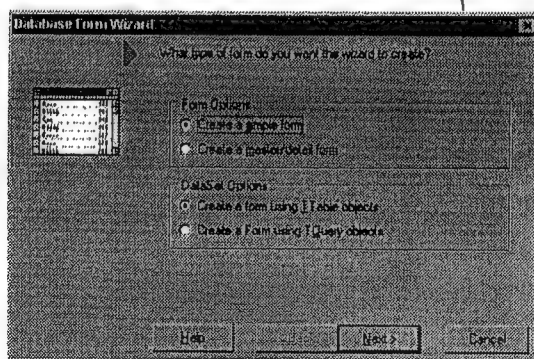
## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

١- اختر File | New... لتبدأ تطبيقاً جديداً. اختر ال Business page tab ، واضغط مرتين ال Database Form Wizard (انظر شكل ١٧-١) .



شكل ١٧-١: ال Database Form Wizard على صفحة ال Business الـ New Items.

٢- تقدم ال Wizard صفحات ال dialog-box ، مع خيارات لإنشاء أنواعاً من ال forms. على الصفحة الأولى ، اختر ال Create a simple form وال Create a form باستخدام TTable (انظر الشكل ١٧-٢) . هذه هي الموصفات الافتراضية حسب النظام .

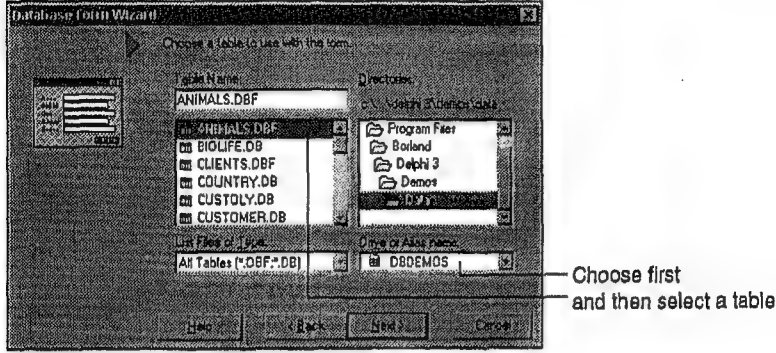


شكل ١٧-٢: الصفحة الأولى الـ TTable.

٣- اضغط زر ال Next لتنتقل إلى الصفحة التالية . (في أي وقت ، يمكنك أن تضغط ال Prev للعودة إلى صفحات سابقة إذا فعلت خطأ ما ، أو إذا غيرت رأيك بشأن إختيارات سابقة) .

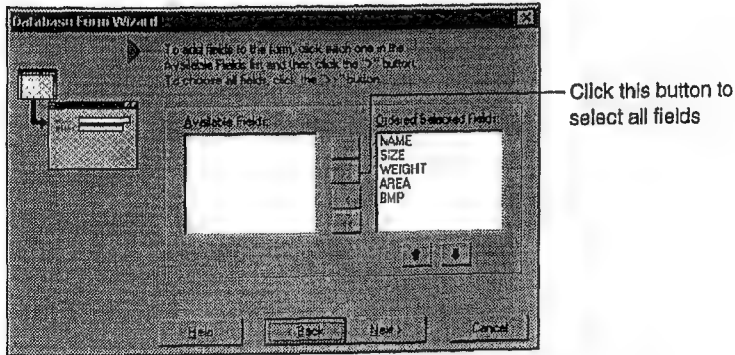


٤- اختر جدولاً بفتح ال ComboBox الذى يحمل labeled Drive أو Alias Name . اختر DBDEMOS أو أي اسم قاعدة بيانات إذا كان متاحاً على نظامك . فى قائمة ال Table Name ، يجب أن ترى الآن أسماء ال Tables التى تكون قاعدة بيانات ال DBDEMOS . اختر ANIMALS.DBF أو Table آخر (انظر شكل ١٧-٣) ثم اضغط Next لتنتقل إلى الصفحة التالية .



شكل ١٧-٣، اختر اسم قاعدة بيانات واسم ال Table باستخدام ال Database Form Wizard .

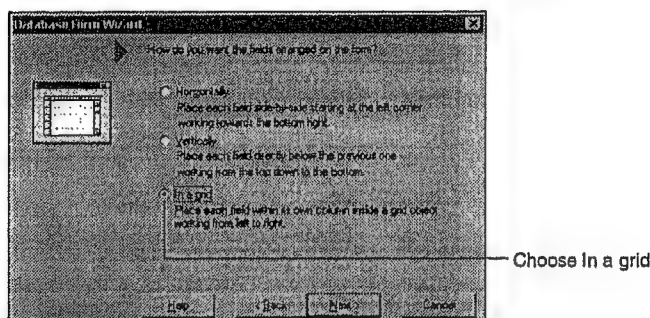
٥- تعطى ال Wizard الآن قائمة بالمجالات المتاحة فى الجدول المختار . اضغط زر السهم المزدوج لتنتقل كل المجالات إلى قائمة ال Selected ، أو يمكنك أن تضغط Shift+click وال Ctrl+click للمجالات الفردية واستخدام أزرار الاسهم الفردية (انظر شكل ١٧-٤) . اسحب واسقط ، أو اضغط الاسهم المشيرة إلى أعلى واسفل ، لترتيب مجالاتك المختارة فى أى ترتيب ، ثم اضغط Next لتنتقل إلى الصفحة التالية .



شكل ١٧-٤، اختر المجالات التى تريد أن تستخدمها .

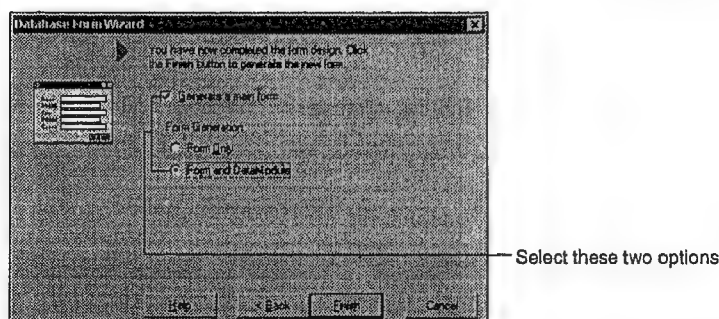
## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

٦- اختر خيار تخطيط : Horizontal لـ side-by-side controls  
data-entry و Vertical لتضع كل control تحت سابقة ، أو Grid لتنشئ مظهراً  
يبدو مثل اللوحة لتصفح وتحرير سجلات متعددة ، واحداً في كل صف . اختر  
In a grid (انظر شكل ١٧-٥) ، ثم اضغط Next لتنتقل إلى الصفحة التالية .



شكل ١٧-٥: اختر خيار تخطيط مثل In a grid.

٧- إن صفحة Wizard الأخيرة تمنحك اختيار إنشاء Form بكل ال components فقط ، أو Form and DataModule ، بال components الغير بصرية في تركيبة البيانات . (هذا يختلف عن ال Wizard السابقين الذين لم يستخدموا data module) . إذا كنت تتابعنا ، تأكد أن ال Generate a main form قد تم اختياره ، واختر خيار ال Form and DataModule (انظر شكل ١٧-٦) . اضغط زر ال Finish (كان يطلق عليه Create فيما سبق) ، لتوليد form قاعدة البيانات الجديدة .

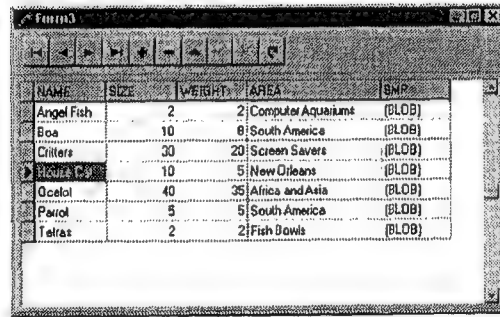


شكل ١٧-٦: هذه هي ال Database Form Wizard dialog page الأخيرة.

**ملحوظة:** فى ال wizard dialog page الأخيرة، إذا قمت بإبطال ال `Generate a main form check box`، فإنك تحتاج أن تدخل برمجة لعرض ال form المؤلدة- باستدعاء ال `ShowModal`، مثلاً، وباختيار `check box`، يتم تصميم ال form الجديدة على أنها ال form الرئيسية للبرنامج، ويمكنك استخدام ال Project Manager لحذف ال form الرئيسية القديمة للمشروع الجديد، والتي تكون خالية ولا تخدم أى هدف.

Note

يمكنك الآن أن تضغط F9 لل compile وتشغيل تطبيق قاعدة البيانات. يوضح شكل ١٧-٧ نافذة البرنامج. بالرغم من أنك لم تضيف أى برمجة، إلا أنه تطبيق قاعدة بيانات بكافة وظائفه! إلى أعلى يوجد ال DBNavigator بأزرار لتصفح السجلات، إضافة صفوف جديدة، حذف سجلات، وما إلى ذلك. إنك حرقى استخدام هذه ال controls، ولكن كن على حذر من أن أى تغييرات تقوم بها يتم تخزينها على الفور فى قاعدة البيانات. على سبيل المثال، قد تغير ال Area for Angel Fish من ال Computer Aquariums إلى ال Tropical Waters.



NAME	SIZE	WEIGHT	AREA	SNIP
Angel Fish	2	2	Computer Aquariums	(BLOB)
Boa	10	8	South America	(BLOB)
Chitras	30	20	Screen Savers	(BLOB)
Chitras C-9	10	5	New Orleans	(BLOB)
Ocelot	40	35	Africa and Asia	(BLOB)
Parrot	5	5	South America	(BLOB)
Tetras	2	2	Fish Dows	(BLOB)

شكل ١٧-٧: تطبيق قاعدة البيانات الذى تم إنشاؤه بواسطة ال Database Form Wizard المعاون والخطوات المذكورة فى هذا الفصل لاسم قاعدة البيانات DBDEMOS وجدول ال Animals.dbf table.

إذا قمت بتجربة الخطوات السابقة، فإن إحدى المشكلات التي قد تلحظها هى أن مجالات بيانات ال bitmap لا يتم عرضها على أنها جرافيك، ولكن بدلاً من ذلك يتم تعريفها على أنها Blobs (Binary Large Objects). هذا يحدث بسبب ال DBGrid، إذا اخترت ال grid التابع لل Wizard، لا يعرف كيف يعرض الجرافيك. إننا نحل هذه المشكلة فيما بعد بادخال ال DBImage فى المشروع. إن ال Wizard لا تقوم بهذا بصورة تلقائية.

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

بالرغم من أن الـ DBNavigator يسهل برمجته ، فإن بعض المستخدمين قد يجدونه معقداً في بادئ الأمر . يأتي مع built-in help hints ، والتي يمكنك تشغيلها بتحديد خاصية الـ ShowHint بـ True . هذا يؤدي إلى تشغيل الـ hints الأصلية ، والتي يمكنك تغييرها إذا لزم الأمر وذلك بتحرير الـ Hints .

يعتبر الـ DBNavigator control مؤثراً ، ويجب أن تكون حذراً من ألا تحذف سجلات أو تقوم بأية تغييرات هائلة ، والتي تصبح دائمة عندما تنقل الحقل الذي يتم إبرازة إلى صف أو سجل آخر . ولكن قبل أن تفعل هذا يمكنك عمل undo لأي تغيير بأختيار زر الـ x . إن ضغط زر علامة الصح يرسل السجل الحالي إلى قاعدة البيانات . وتقوم أزرار الزائد والناقص بإضافة وحذف سجلات . والسهم المستدير يجدد البيانات باعادة التحميل من الجدول . وتقوم الأزرار الأخرى بالتصفح عبر سجلات الجدول .

**Tip فكرة :** لإدخال forms قاعدة بيانات إضافية في تطبيق ، اختر أمر الـ DatabaselForm Wizard . هذا يؤدي إلى تشغيل نفس الـ Wizard الذي اخترته بالـ FileNew... في الـ New Items Business page .

## الـ components ومصطلحات قاعدة البيانات:

قبل إنشاء قواعد بيانات وكتابة تطبيقات للتوصل إلى معلومات قواعد البيانات ، فمن المهم أن تفهم ثلاث مصطلحات متعلقة ببعضها عند استخدامها في برمجة قاعدة البيانات وبواسطة الـ BDE . وهذه المصطلحات هي :

● **Table-** هذا هو مصدر بيانات في ملف مسطح فردي ، والذي تتصوره على أنه يحتوي على صفوف (سجلات) وأعمدة (مجالات) . وأحد هذه المجالات في الـ Table هو المفتاح الأساسي الذي يتم فهرسة المعلومات الموجودة في الملف المسطح داخله . يمكن أيضاً فهرسة الـ Tables في مفاتيح ثانوية . ويطلق عادة على الـ Table اسم الـ dataset .

● **Query-** مثل الـ Table ، بالرغم من أنه يمكن استخدام الـ Table للتوصل إلى بيانات الـ SQL ، إلا أن استخدام الـ Query يبسط هذه العملية ، وخاصة عندما تأتي البيانات من الـ SQL server . يمكنك أيضاً استخدام الـ Query لإنشاء

إتحادات منطقية من مجموعات البيانات الغير متشابهة (لا تستطيع الـ Tables فعل هذا)، ويمكنك أن تشترك في بيانات من مصادر مختلفة مثل الـ Paradox Tables والـ SQL.

● **Database** - هذه هي مجموعة من Table واحد أو أكثر (اثنين على الأقل دائماً). عندما يشير أحد الـ Tables إلى آخر من خلال قيمة مفتاح في حقل محدد، وتعرف النتيجة على إنها قاعدة بيانات علاقية.

● **Alias** - هذا هو اسم مسجل مع الـ BDE الذى يخفى اسماء المسارات ومحركات القرص التى تحدد موضع ملفات قاعدة البيانات. استخدم دائماً أسماء مستعارة للإشارة إلى قواعد البيانات، لا تضع ملف Code وأسماء مسارات فى تطبيقاتك باستخدام الأسماء المستعارة، يمكنك أن تنقل ملفات قاعدة البيانات الخاصة بك إلى مواضع أخرى- أو تنقلها إلى دليل على Grid- وتعمل جميع تطبيقاتك بلا تعديل.

### إنشاء قاعدة بيانات جديدة:

بالرغم من أنه يمكن استخدام Delphi لكتابة تطبيق يمكنه إنشاء قواعد بيانات جديدة، فإن المبرمجين الحاذقين للـ Inprise Corporation قد قاموا بهذا العمل لك فى الـ Database Desktop، وهو متوفر مع Delphi. استخدم هذه المنفعة لإنشاء جداول قاعدة بيانات جديدة، لتعديل البيانات فى قواعد البيانات الموجودة، لرؤية بيانات قاعدة البيانات، ولإنشاء اسماء مستعارة لمصادر البيانات. إن طبقات الـ Professional والـ Client-Server لـ Delphi توفر أيضاً Data Base explorers يمكنك استخدامها لتصفح الـ Tables وهياكلها. إذا كان لديك إحدى هذه الطبقات من Delphi، اختر الـ plorers من قائمة الـ Database. يوضح شكل ١٧-٨ الـ Database Desktop وهو يتصفح DBDEMOS Table Customer.db. استخدم الـ Database Desktop، المتوفر مع جميع طبقات Delphi، لإنشاء وتعديل Tables قواعد البيانات فى forms متنوعة. وتوفر أيضاً بعض طبقات Delphi utility explorers قاعدة البيانات مثل الـ SQL Explorer. يوضح شكل ١٧-٩ الـ SQL Explorer وهو يتصفح هيكل نفس الـ Table المستخدم فى شكل ١٧-٨. (يتوفر الـ SQL Explorer فقط مع طبعة الـ

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

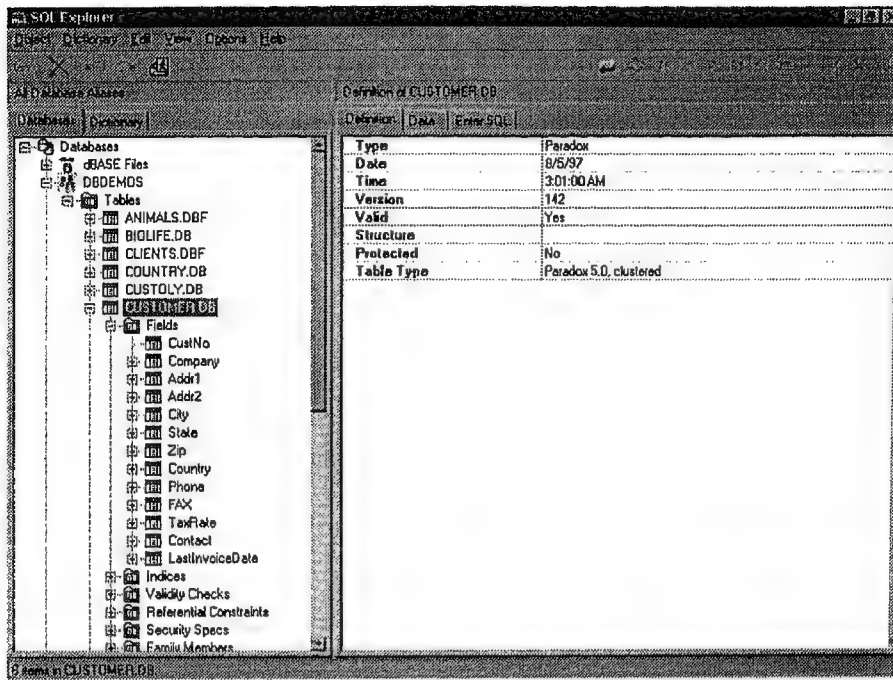
Client-Server من Delphi، تتضمن طبقات ال Professional نوع Explorer مشابه ولكن بدون امكانيات ال (SQL).

City	Company	Address	City	State
1. 221.00	Kaial Dine Shoppes	4975 Sugallol Hwy	Suite 103	Kapaa Kapa
2. 221.00	Unicop	P.O. Box 2647		Frederick
3. 361.00	Sight Diver	1 Neptune Lane		Kota Papias
4. 364.00	Cayman Dine World Unlimited	P.O. Box 641		Grand Cayman
5. 366.00	Tom Sawyer Dine Center	632-1 Third Frydenhof		Christiansburg
6. 1200.00	Blue Jack Aqua Center	23-730 Paddington Lane	Suite 310	Waipahu
7. 1304.00	VIP Divers Club	32 Main St.		Christiansburg
8. 1510.00	Ocean Paradise	P.O. Box 8745		Kailua-Kona
9. 1513.00	Fantastique Aquatics	Z12 999 #12A-77 A.A.		Bogota
10. 1561.00	Mamot Divers Club	872 Queen St.		Kitchener
11. 1560.00	The Depth Charge	15243 Underwater Fwy		Marathon
12. 1563.00	Blue Sports	203 12th Ave. Box 745		Gibbsville
13. 1624.00	Makal SCUBA Club	P.O. Box 8534		Kailua-Kona
14. 1645.00	Action Club	P.O. Box 6451-F		Sarasota
15. 1651.00	Jamaica SCUBA Centre	P.O. Box 68		Negril
16. 1660.00	Island Fritata	6133 173 Stone Avenue		St. Simons Isle
17. 1684.00	Adventure Undersea	P.O. Box 744		Belize City
18. 2118.00	Blue Sports Club	63306 Naz Perce Street		Largo
19. 2135.00	Frank's Dine Supply	1458 North 44th St.		Eugene
20. 2156.00	Dave Jones Locker	245 South 18th Place		Vancouver
21. 2163.00	SCUBA Heaven	P.O. Box Q-8674		Nassau
22. 2165.00	Shang-Ha Sports Center	P.O. Box D-6496		Frederick
23. 2315.00	Dive's of Confu, Inc.	Mamcoat Place 54		Ayios Mathaios
24. 2354.00	Kirk Enterprises	42 Aqua Lane		Houston
25. 2975.00	George Bean & Co.	403 King Salmon Way		Lugoff
26. 2904.00	Professional Divers, Ltd.	4704 Melinda St.		Hoover
27. 3041.00	Divers of Blue-green	634 Complex Ave.		Pelham

شكل ١٧-١٨ يوضح هذا الشكل ال Database Desktop وهو يتصفح محتويات ملف ال DBDEMOS Customer.db المتوفر مع Delphi.

**ملحوظة:** بعد ال Database Desktop تطبيق مستقل - يمكنك تشغيله من قائمة ال Tools الخاصة ب Delphi أو من دليل تركيب Delphi. يعتبر ال SQL Explorer مكمل ل Delphi يمكن تشغيله فقط باختيار أمر ال DatabaseExplore.

**على القرص المدمج:** لإنشاء قاعدة بيانات جديدة، فإن أول خطوة في تحديد مكان لتخزين معلوماتك، وتعيين اسم مستعار يشير إلى هذا الموضوع. لديك اختيارين: يمكنكم استخدام الملفات الموجودة على القرص المدمج بهذا الكتاب. لاستخدام الملفات، اتبع الخطوات المرقمة التالية. لإنشاء قاعدة بيانات جديدة خالية، إنتقل إلى "إنشاء قاعدة بيانات ال Wines".



شكل ٩-١٧: هذا الشكل يوضح الـ SQL Explorer وهو يتصفح هيكل ملف الـ DBDEMOS Customer.db المتوفر مع Delphi.

### استخدام قاعدة بيانات الـ Wines:

على القرص المدمج: إتبع هذه الخطوات لإستخدام ملفات قاعدة بيانات الـ Wines الموجودة على القرص المدمج في دليل الـ Source\WinesDemo



١- انسخ دليل الـ Source\Data\Wines وملفَيّـة، Wines.px وWines.db، على دليل جديد يسمى C:\Database\Wines. يمكنك استخدام اسم مسار وحرف محرك مختلفين، ولكن الدليل الداخلي يجب أن يسمى Wines.

٢- ابدأ الـ Database Desktop باستخدام الـ Windows Explorer أو Taskbar، أو باختيار أمر الـ Tools|Database Desktop التابع لـ Delphi عندما تظهر نافذة الـ Database Desktop، اختر أمر الـ Tools|Alias Manager.

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

٣- اضغط زر الـ Next لإنشاء اسم مستعار جديد . أدخل الاسم المستعار WINES ، وتأكد من أن الـ Driver Type محدد بـ STANDARD (الافتراضي) .

٤- اضغط زر الـ Browse ، وفي الـ Directory Browser dialog الناتج ، اختر حرف المحرك والأدلة لتحديد موضع مسار الـ C:\Database\Wines (أو يمكنك كتابة اسم المسار في الـ Directories editor) . اضغط OK لتغلق الـ Directory Browser dialog .

٥- اضغط Keep New ثم اضغط OK لإغلاق نافذة الـ Alias Manager . عندما تُسأل ما إذا كنت سوف تحفظ الـ Public Aliases في IDAPI32.CFG ، أجب بـ Yes .

**ملحوظة:** إن الفصل القادم يصف كيفية إنشاء قاعدة بيانات Wines خالية وجديدة . إذا إتبع الخطوات السابقة ، إنتقل إلى " Components قاعدة البيانات " .

Note

### إنشاء قاعدة بيانات الـ Wines

توضح النقاط التالية كيفية إنشاء قاعدة بيانات Wines جديدة وخالية . وهذا يساعدك على معرفة كيفية استخدام الـ Database Desktop لإنشاء Tables قاعدة البيانات الخاصة بك .

إن الخطوة الأولى هي إنشاء مكان لتخزين الملفات . إن تحديد اسم الدليل والملف بدقه أمر يرجع إليك ، ولكن على نظامي ، فقد استخدمت الـ Windows Explorer لإنشاء دليل ، D:\Database\Wines (يمكنك استخدام حرف محرك مختلف وكذلك اسم مسار مختلف إذا أردت ذلك) . بعد إنشاء الدليل ، اتبع هذه الخطوات لإنشاء اسم مستعار لمسار قاعدة البيانات ، ولإنشاء ملفات الـ Table :

١- إبدأ Database Desktop باستخدام الـ Windows Explorer أو الـ Taskbar ، أو باختيار أمر الـ Tools\Database Desktop التابع لـ Delphi . عندما تظهر نافذة الـ Database Desktop ، اختر أمر الـ Tools\Alias Manager .

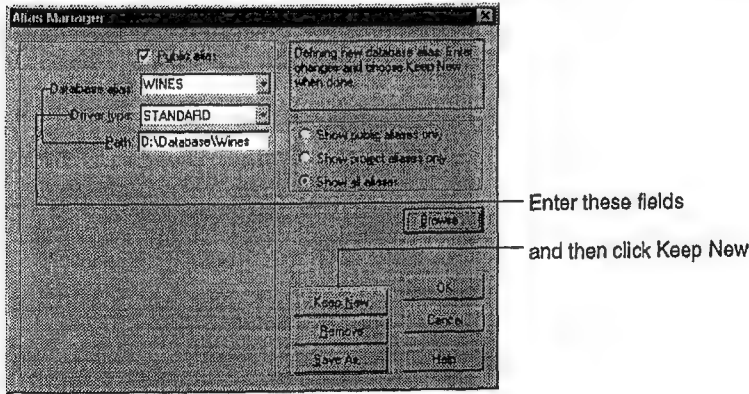


٢- اضغط زر الـ Next لإنشاء اسم مستعار جديد . أدخل الاسم المستعار- يمكن أن يكون string ، ولكنه عادة يكون مثل ملف الـ Grid أو الدليل الذى تكون قاعدة البيانات موضوعة فيه . على سبيل المثال ، أدخل WINES كاسم مستعار لقاعدة بيانات قد تستخدمه لإقتفاء أثر قائمة جرد مخزن خمور .

٣- حدد الـ Driver Type بـ STANDARD (وهى القيمة الافتراضية حسب النظام) . إلا إذا كنت تنشئ قاعدة بيانات الـ server والعميل ، ففى هذه الحالة يمكنك اختيار INTRBAS .

٤- اضغط زر الـ Brows ، واسحب إلى أسفل الـ "Drive or Alias ComboBox control" . اختر D: (أو حرف المحرك الذى استخدمته لإنشاء دليل الـ Wines) ، ثم اضغط مرتين Database يتبعها Wines فى قائمة المسار أعلاه . يجب أن يقرأ حقل الـ Directories هذا "D:\Database\Wines"

٥- اضغط زر الـ OK لإغلاق متصفح الدليل . يوضح شكل ١٧-١٠ عرض الـ Database Desktop فى هذه المرحلة . اضغط OK وأجب بـ Yes عندما تسأل عما إذا كنت سوف تحفظ الـ Public Aliases فى IDAPI32.CFG يمكنك الآن أن تشير إلى اسم الـ WINES OLE فى تطبيقات Delphi بدلاً من محرك الـ code الصلب واسم المسارات فى قاعدة البيانات هذه .



شكل ١٧-١٠: مسار واسم الـ alias WINES مسجلين بواسطة الـ Database Desktop.

إن تسجيل اسم الـ Wines OLE وإنشاء مكان لتخزين معلومات قاعدة البيانات . لفعل هذا، إتبع الخطوات التالية، التى تنشئ قاعدة بيانات فى ملف مسطح لقائمة جرد بسيطة لمخزن خمور:

١- إبدأ Database Desktop إذا لم يكون عاملاً بالفعل .

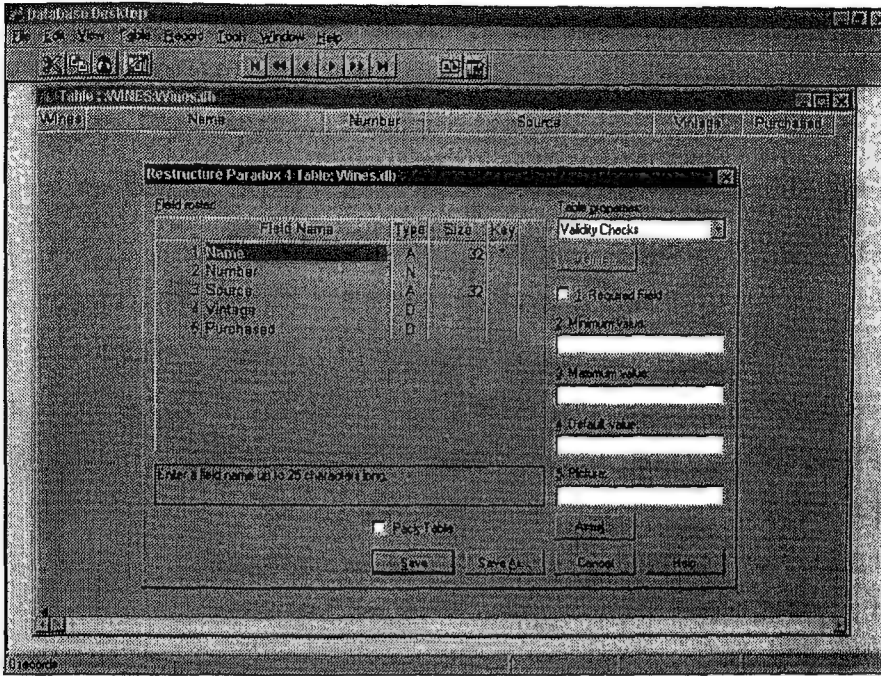
٢- اختر FileNew واختر أمر القائمة الفرعية Table... . إنك ترى قائمة forms قاعدة البيانات المتاحة . اختر Paradox 7، وهو الافتراضى . (عند إنشاء قاعدة بيانات جديدة، يعتبر Paradox هو أفضل اختيار- فهو يوفر أغلب أعداد أنواع بيانات المجالات وكذلك إمكانيات الفهرسة الرئيسية المعقدة . بالطبع، يمكنك اختيار أى form قاعدة بيانات أخرى مثل الـ dBASE أو الـ Intrbase إذا أردت) .

٣- إنك ترى الآن شاشة الإدخال الرئيسية لإدخال وتحرير المجالات . أدخل أسماء المجالات ، أنواعها (اضغط قضيب المسافة إذا كانت قائمة)، وحجم الحقول الأبجدية، ونجمة (يمكنك كتابة أى رمز) للإشارة إلى حقل Primary key . يوضح الجدول ١٧-١ قائمة ببعض المجالات (هذا هو مجرد عرض، وقاعدة بيانات مخزن الخمور الحقيقية يكون لها مجالات أكثر من هذه بكثير) . يوضح شكل ١٧-١١ هيكل Table قاعدة البيانات التام فى الـ Database Desktop .

٤- بعد إدخال حقولك، اضغط زر الـ Save As . ادخل Wines فى مربع تحرير Filename، ثم اختر Alias للإشارة إلى Table قاعدة البيانات الجديدة . اختر Wines Alias الذى أنشأته سابقاً (Wines)، ثم اضغط Save لإنشاء الـ Tables وأي ملفات متعلقة به مثل الفهارس المعتمدة على برنامج تشغيل قاعدة البيانات الذى اخترته فى الخطوة رقم (٢) . إذا كنت مستمر فى المتابعة، يجب أن يكون لديك الآن ملفين، Wines.db و Wines.px فى دليل الـ \Database\Wines .

## الـ Components قاعدة البيانات:

بعد إنشاء قاعدة البيانات الجديدة أو اختيار واحدة موجودة بالفعل تكون قد سجلت لها alias، فأنت مستعد لكتابة تطبيق Delphi لإدخال، تحرير، ورؤية معلومات قاعدة البيانات . كما ذكر، إحدى فى الطرق للبدء هى استخدام الـ Database Form Wizard (استخدم الـ FileNew....، اضغط الـ Business page tab، واضغط مرتين الـ Database Form Wizard .



شكل ١٧-١١: هيكل Table قاعدة بيانات Wines.db التام في ال Database Desktop.

جدول (١٧-١) حقول قاعدة بيانات مخزن الخمر العتيقة.

اسم الحقل	النوع	الحجم	المفتاح
Name	Alpha	32	*
Number	Number		
Source	Alpha	32	
Vintage	Date		
Purchased	Date		

ولكن، كما سأوضح فيما بعد، يمكنك إنشاء تطبيقات قاعدة بيانات من لاشئ وذلك بإضافة component على form. فلا يتعين عليك أن تستخدم ال Wizard. إن أداء هذه الخطوات بنفسك أيضاً يعلمك بما تفعله component قاعدة

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

البيانات المتنوعة وكيف تتفاعل مع بعضها . حتى إذا خططت لاستخدام ال Database Form Wizard ، فيجب أن تفهم هذه العلاقات حتى تقوم ببرمجة تطبيقات قاعدة البيانات بنجاح .

**Tip** **فكرة:** لأن Delphi يولد ملفات متعددة لا تحتاج إلى أن تنشرها على المستخدمين النهائيين مع تطبيقاتك التامة ، فمن الأفضل أن تنشئ أدلة منفصلة لقواعد البيانات وتطبيقاتها .

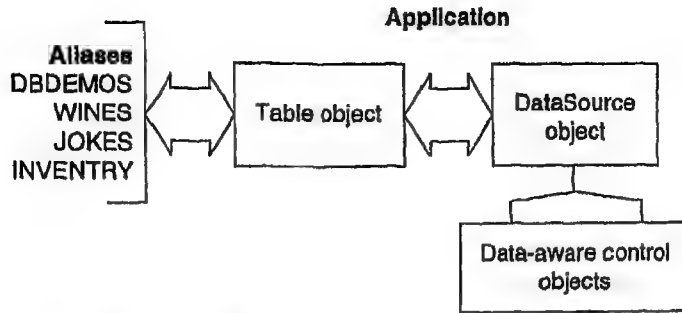
### ال Data Access components:

إن ال components الموجودة في ال Data Access palette توفر وصولاً إلى Tables قاعدة البيانات . تصور ال Data Access على أنها "بوابات" لمعلومات قاعدة البيانات . في أغلب الأحيان ، إنك تحتاج نماذج من Table و DataSource . وتؤدي ال Data Access الأخرى عمليات SQL مثل (Query) ، وتنفذ عمليات عامة مثل تحديث كل الحقول في سجلات مناسبة (BatchMove) . ويتم استخدام ال Database وال Session بصورة تلقائية عند الحاجة لتنظيم الوصول إلى Tables قاعدة البيانات - إنك لن تحتاج هذه ال components إلا إذا كنت تنشئ برمجيات متقدمة لأداء مهام مثل الوصول إلى controls إتصال متعددة . إنك لن تحتاج غالباً إلى StoredProc أو ال Update SQL إلا إذا كنت تطور قواعد بيانات ال server والعمل .

**ملحوظة:** إن النسخ الأولى من Delphi تضمنت برنامج ال ReportSmith وال Report component لتوليد تقارير قاعدة بيانات . ولقد تم استبدال هذه البنود ب components توليد التقارير الواسعة النطاق على لوحة ال QReport . انظر الباب التالي لمزيد من المعلومات عن هذه ال components وعن ال TDBChart أيضاً .

إن أول component تدخله في form يكون عادة هو ال Table . وهذا ال object ينشئ كوبرى بين التطبيق و alias قاعدة البيانات كما هو موضح في شكل ١٢-١٧ . بالإضافة إلى ال Table ، فإنك تحتاج إلى ال DataSource ، الذي يربط ال data-aware إلى قاعدة البيانات . إن ال DataSource يغذى بالبيانات من وإلى

objects أخرى وال Table . ويتولى ال Table التبادلات الفعلية لمساعدة البيانات .



شكل ١٢-١٢: يكون ال Table كوبرى بين التطبيق وقاعدة البيانات، والمعروفة باسم alias مسجل. يربط ال DataSource بين ال Table وال data-aware مثل ال DBNavigator وال DBEdit.

وهذه الأعمال بين ال Table، وال DataSource، وال data-aware، تقع فى ال Borland Database Engine (BDE)، الذى يؤدى العمل الحقيقى من قراءة وكتابة بيانات فى ال form المناسبة. ويمكن لل data-aware أن تربط. إضافياً لإنشاء شاشات إدخال بيانات متفاعلة. على سبيل المثال، إن ربط DBNavigator بال DataSource، والذى هو مربوط بال Table، يؤدى إلى إنشاء toolbar متصفح يمكنك استخدامه لرؤية، تحرير، ادخال، وحذف سجلات معروضة فى ال DBEdit ونوافذ control أخرى. وإلى أن تعتمد على أى ال objects ترتبط بال objects الأخرى، فقد تبدو هذه العلاقات معقدة بعض الشيء ولكن، بعد أن تنشئ واحداً أو اثنين من التطبيقات، فستجد أن شاشات إدخال البيانات تتجانس مع بعضها مثل المكعبات الخاصة بالأطفال.

إتبع هذه الخطوات لإنشاء Table، و DataSource، و data-aware لإدخال ورؤية معلومات فى قاعدة بيانات مخزن الخمر (يمكنك أيضاً استخدام هذه التعليمات مع أى قاعدة بيانات أخرى على نظامك):

١ - إبدأ تطبيقاً جديداً باختيار ال Database Form Wizard-I - فقد أنشأت ال code اللازمة بالطريقة الصعبة.

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

٢- ادخل Table و DataSource من الـ Data Access palette فى form . يتم عرضهما كأيقونات لا تظهر فى وقت التشغيل . لقد استخدمت الـ Names الافتراضية لـ Delphi لكل الـ objects . فى هذا العرض ، ولكن فى برامجك ، قد تريد أن تعين خصائص Name واصفة للـ objects . (إننى أؤيد استخدام الكلمات Table و Source فى هذه الأسماء . على سبيل المثال ، MasterTable ، NamesSource تعتبر اسماء objects جيدة) .

**Tip** **فكرة :** يمكنك وضع components غير بصرية مثل الـ Table والـ DataSources فى تركيبة بيانات بدلاً من نافذة form مرئية . يمكنك أيضاً استخدام data moules لإنشاء تركيبات عامة للتوصل لقاعدة البيانات تستطيع التطبيقات المتعددة التشارك فيها . انظر ' استخدام data moules object ' فى هذا الباب لمزيد من المعلومات .

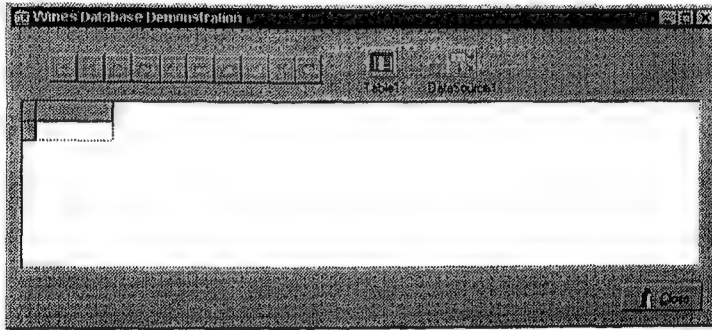
٣- اختر Table1 object ، فى الـ Properties الخاصة بالـ Object Inspector ، اضغط السهم الواقع بعد خاصية الـ DatabaseName . هذا يسقط قائمة بأسماء قواعد البيانات وأسماء Alias المسجلة لها . اختر WINES أو اسم آخر وارد فى القائمة .

٤- تذكر أن قاعدة البيانات قد تتكون من Table واحد أو أكثر ، وتسمى أيضاً بمجموعات البيانات . بالرغم من أن قاعدة البيانات لمخزن الخمور الخاصة بنا لها Table مجموعة بيانات واحد فقط ، بالإضافة إلى اسم Alias للـ DatabaseName ، فعليك أيضاً أن تحدد الـ TableName . اختر خاصية الـ TableName للـ Table1 ، واختر Wines.db أو اسم Table آخر . تذكر : كل Table objects يجب أن تحدد خصائص الـ DatabaseName والـ TableName لها .

٥- بعد ذلك ، اختر DataSource1 . اربطه بالـ Table1 باختيار Table1 من قائمة اللائحة لخاصية الـ DataSet . بهذه الخطوة تكون قد أتممت الحد الأدنى للمتطلبات لربط التطبيق بقاعدة بيانات ، ويمكنك الآن أن تدخل الـ data-aware فى الـ form لرؤية وتحرير معلومات قواعد البيانات . تذكر : كل الـ DataSource يجب أن ترتبط من خلال خاصية الـ DataSet بـ Table .

٦- وأحد ال component التى تستخدمها غالباً هو ال DBNavigator. اختر ذلك ال component من ال Data Controls palette، أضيف إلى ال form فى موضع ملائم، فى مكان ما بالقرب من الحد العلوى. خصص ال DataSource1 لخاصية ال DataSource التابعة لل DBNavigator. هذا يخبر ال data-aware أين تجد بياناتها. ولأن مجموعة البيانات لا تكون مفتوحة فى هذا الوقت، فإن تحديد خاصية ال DataSource يجعل أزرار ال DBNavigator غير عاملة. تذكر: كل ال data-aware يجب أن ترتبط عبر خاصية ال DataSource بـ DataSource.

٧- لكى توفر سطح لرؤية وتحرير معلومات قاعدة البيانات، أضيف BGrid object من ال Data Controls palette على ال form. حدد ال DataSource1 التابعة لل DBGrid. أعد تحرير حجم وموضع ال controls والنافذة كما تريد. يوضح شكل ١٧-١٣ نافذة form التطبيق المتطور فى هذه المرحلة فى Delphi.



شكل ١٧-١٣، ال form المتطورة توضح الحد الأدنى من ال components المطلوبة لتطبيق قاعدة بيانات نموذجى.

٨- والآن استعد لبعض Wizard. اختر Table1، واضغط مرتين خاصية ال Active لتغيير قيمتها إلى True. هذا يفتح Table قاعدة بيانات داخل التطبيق المتطور- لا يجب عليك أن تقوم بتشغيل البرنامج. إنك ترى أسماء حقول ال Table قاعدة البيانات فى ال DBGrid، وإذا كان Table به أية معلومات، فإنك تراها أيضاً. يعتبر ال DBGrid الآن ال active control، وتكون قاعدة البيانات مفتوحة فى هذا الوقت. من الملائم أن يكون لديك بيانات جديدة متاحة عند تصميم تطبيقات قاعدة البيانات- على سبيل المثال، يمكنك بسهولة أكثر أن تجد حجم

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

components لتعرض معلومات Table - ولكن قد تحدد الـ Active بـ False لإغلاق Table. فى هذه الحالة، يمكنك إدخال التعيين التالى فى OnCreate الخاص بالـ form (أو فى Procedure آخر) لفتح قاعدة البيانات والـ Table فى وقت التشغيل :

```
Table1.Open; // Open database and table
```

**تحذير:** لا تستطيع الـ Database Desktop أن تعيد هيكلة Table مفتوح. إذا لم تستطع فتح Table باستخدام الـ Database Desktop، إرجع إلى Delphi وحدد حقل الـ Active الخاص بالـ Table بـ False. أو، احفظ وأغلق تطبيقك، ثم حاول استخدام الـ Database Desktop مرة أخرى.



٩- يوضح شكل ١٧-١٤ تطبيق الـ WinesDemo أخرى.

**على القرص المدمج:** فى كثير من الحالات، تستخدم الـ DBGrid لرؤية وتحرير معلومات Table قاعدة بيانات. ولكن هذه ليست هى الطريقة الوحيدة لفعل هذا- يمكنك أيضاً إنشاء forms إدخال بيانات باستخدام العديد من الـ data-aware. على سبيل المثال، باستخدام الـ DBGrid، يمكنك إنشاء form إدخال قاعدة بيانات توضح سجلاً واحداً فى كل مرة. يوضح شكل ١٧-١٥ ترتيب واحد ممكن للـ control بـ Table الـ Wines.db. يوجد التطبيق التام على القرص المدمج فى دليل Source\WinesEntry - افتح ملف مشروع الـ WinesEntry.dpr هذا الدليل راضع برمجة مستخدم، وهو غير وارد هنا).



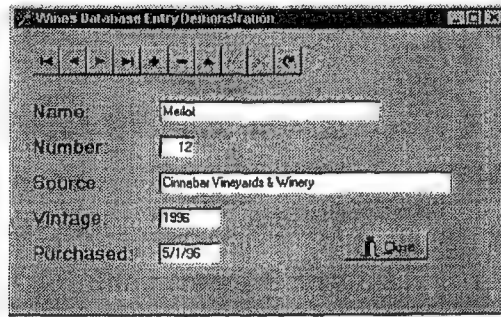
Name	Number	Source	Year	Purchased
Chardonnay	8	Mission View Estate Vineyard	1995	6/1/95
Merlot	12	Cinnabar Vineyards & Winery	1996	5/1/96
Sauvignon Blanc	12	Topolos Winery	1996	3/15/97
Cabernet Sauvignon	10	Bales Ranch Santa Cruz Mountains	1992	2/2/93

شكل ١٧-١٤: تطبيق الـ WinesDemo المكتمل.



لإنشاء برنامج ال WinesEntry الظاهر فى الشكل، فقد أضفت Table DataSource لـ forms جديدة. ثم اخترت ال Table، وحددت ال WINES alias لخاصية ال DatabaseName وخاصية ال TableName التابعة له بـ Table1. بـ Wines.db. لقد اخترت ال DataSource وحددت خاصية ال DataSet التابعة له بـ Table1. بعد هذه الخطوات، فقد أضفت DBNavigator وخمسة DBEdit مع ال Labels و Button كما هو موضح فى شكل ١٧-١٤. إن حقل ال DataSource الخاص بال DBNavigator وال DBEdit يكون محدداً بـ DataSource1. تكون خاصية ال DataField لكل DBEdit محددة باسم حقل مختلف فى جدول ال Wines.db.

أخيراً، لقد اخترت ال Table1 وغيّرت خاصية ال Active التابعة له إلى True. ثم ضغطت F9 لتشغيل البرنامج التام.



شكل ١٧-١٥: بدلاً من ال DBGrid، هذه النسخة من تطبيق قاعدة بيانات ال Wines تستخدم حقول ال DBEdit لتحرير وعرض سجلاً واحداً فى المرة.

### ،Data-aware Controls

عند تصميم forms إدخال البيانات، يجب أن تعرف أى المجالات تريد أن تعرضها وفى أى موضع. لديك إختياران أساسيان: يمكنك استخدام ال DBGrid لعرض صفوف وأعمدة، أو يمكنك استخدام controls منفرد مثل ال DBText، DBEdit، و DBMemo لحقول مختارة. يعرض ال DBGrid كل الحقول المتاحة فى ترتيبها المادى حسب النظام الافتراضى للبرنامج. وإن أمر عرض أى الحقول المنفرد ال Data-aware الأخرى يرجع إليك.

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

ولكن، فى كثير من الأحوال إنك تريد أن تحد من الحقول التى تعرضها الـ DBGrid، أو قد تريد أن تغير ترتيب أعمدها. بالإضافة إلى ذلك، قد تريد أن تنشئ حقلاً أو calculated field. هو حقل وهمى يعرض نتيجة صيغة باستخدام قيم فى خلايا أخرى. يمكنك تنفيذ هذه المهام بتعديل الحقول الفعلية التى يوفرها الـ Table component من المجالات المادية لمجموعة البيانات. يمكن أن يعرض المجال الفعلى جزءاً أو نسخة معدلة من المجال المادى - على سبيل المثال، الاسم الأخير لشخص.

لأداء هذه الأنواع من الأعمال، فبدلاً من استخدام الحقول المادية فى جدول، يمكنك أن تأمر الـ Table أن ينشئ مجموعة من objects المجالات التى فى الحقيقة، تحتوى على أعمدة مجموعة البيانات المادية. تستطيع بعض الأعمدة مباشرة أن تنتقل من العمود المادى إلى الغطاء - يمكنك مثلاً، أن تخبر الـ Table أن يستخدم حقل الـ Name كما هو موجود فى مجموعة البيانات. والأعمدة الأخرى يمكن أن تكون على حسب ما تريده أنت. وهذه الأعمدة "الوهمية" قد تحسب قيم مؤسسة على مجالات أخرى، أو إنها قد تعرض معلومات مهيئة بطريقة خاصة باستخدام البيانات الخام من الـ Table.

**على القرص المدمج:** حاول تجربة هذه المفاهيم باستخدام قاعدة بيانات

الـ Wines من الفصل السابق - أو يمكنك استخدام Database Form

Wizard لإنشاء تطبيق لـ Table مجموعة بيانات. لتحرير حقول الـ

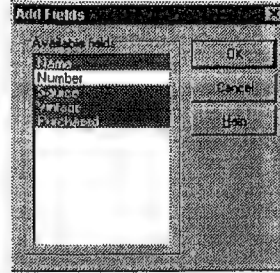


Table، اضغط مرتين الـ Table1 فى الـ form.

(إن الملفات الكاملة لهذا الفصل توجد على القرص المدمج فى دليل الـ Source\WinesCalc). كما هو موضح فى الشكلين ١٧-١٦ و ١٧-١٧، بالضغط مرتين على الـ Table تعرض محرر الـ Fields الخاص بالـ component والذى يبدأ كنافذة خالية تحمل فى هذه الحالة الـ Mainform.Table1. اضغط يميناً بينما تشير بال مؤشر إلى هذه النافذة. هذا يظهر قائمة ذات أمرين عاملين:

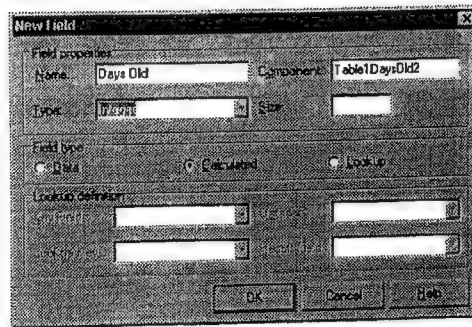
● **Add Fields** - استخدم هذا الأمر لإضافة أعمدة مجموعة بيانات مادية إلى الـ Table. على سبيل المثال، إذا أردت استخدام حقول Name كما هو فى ملف مجموعة البيانات، استخدم الـ Add Fields لاختياره. إذا لم تضف حقلاً، فيكون

مخفياً فى المخرجات النهائية- فى ال DBGrid ، مثلاً. كل حقل مضاف يتم تمثيلة ك object فى ال form class . يوضح شكل ١٧-١٦ ال Add Fields dialog .



شكل ١٧-١٦، ال Add Fields dialog .

• **New Field** - استخدم هذا الأمر لإنشاء حقول وهمية جديدة، والتي يمكن أن تكون واحدة من ثلاثة أنواع: Data ، Calculated ، أو Lookup . يتم نقل ال Data fields من البيانات فى ملف dataset . على سبيل المثال، يمكنك إنشاء حقل بيانات لعرض uppercase strings فى حروف كبيرة وصغيرة. وال Calculated fields . تطلق event handle الذى يمكنك استخدامها لأداء الحسابات . (أو أى عمليات أخرى) باستخدام قيم حقول أخرى . فالحقل المحسوب، مثلاً، قد يعرض عدد الأيام المنقضية بين تاريخين . والنوع الثالث من ال New Field الذى يمكنك إنشاؤه يدعى ال lookup field . هذا يحصل على بيانات من مجموعة بيانات أخرى- اسم شركة، مثلاً، يعطى للعميل رقم ID . كل حقل جديد يتم تمثيلة ك object فى ال form class . يوضح شكل ١٧-١٧ ال New Field dialog .



شكل ١٧-١٧، ال New Field dialog .

لإنشاء برنامج العرض WinesCalc ، الذى يوضح كيفية إنشاء calculated fields ، قمت بالضغط مرتين على الـ Table1 فى نسخة من تطبيق الـ WinesDemo ، ثم استخدمت أمر الـ Add Fields لإضافة أعمدة الـ Name ، والـ Source ، والـ Vintage ، والـ Purchased إلى شكل الـ Table1 . لأننى لم اختر حقل الـ Number فهو مختفى فى الـ Grid النهائية (انظر شكل ١٧-١٦) .

ثم استخدمت أمر الـ New Field لإنشاء الـ Days calculated field ، من نوع الـ Integer . ففى البرنامج ، يتم تمثيل هذا الحقل على أنه object الـ Table1DaysOld من نوع الـ TIntegerField . (انظر شكل ١٧-١٧) .

**ملحوظة:** إن التغييرات التى قمت بها فى محرر الـ Fields لا تؤدي إلى تعديل مجموعة البيانات المادية ، لذلك لا تأس أو تغضب من التجربة . استخدم زر الـ Add لكى تعيد أى حقول تكون قد حذفته . (جرب هذا : اضغط Clear all ، وأجب بـ yes على الاستفسار الذى يسألك إن كنت تريد أن تحذف كل الحقول . ثم اضغط Add و OK لكى تعيد إضافة الحقول مرة أخرى . كما ترى ، إن المحذوفات لا تكون دائمة) . يتم تعريف كل مجال فى الـ form على أنه component من نوع حقل مجموعة البيانات - TFloatField ، TStringField ، TDateField ، وغيرها . إن استخدام محرر الـ Field يعيد برمجة تعريفات هذه الـ objects .

إن كل حقل فى مجموعة بيانات الـ Table يصبح object فى الـ form class . استخدم الـ Object Inspector لتعديل خصائص هذه الـ objects بنفس الطريقة التى برمجتها بها قيم الـ objects أخرى . على سبيل المثال ، اختر الـ Table1 فى قائمة لائحة الـ Object Inspector ، وقيم بتغيير خاصية الـ ReadOnly للـ object تصبح True . وعندئذ يصبح المستخدمون غير قادرين على عمل أية تغييرات للقيم فى هذا الحقل .

**فكرة:** لاختيار حقل ، اضغط مرتين الـ Table component على الـ form ، واختر الـ object بواسطة اسم العمود الخاص به . أو يمكنك اختيار الـ object باستخدام قائمة لائحة الـ Object Inspector . بأى

الطريقتين، فبعد اختيار الـ object، استخدم نافذة الـ Object Inspector لتعديل خصائص الحقل.

يقوم محرر الـ Field ببرمجة عدد، وأسماء، وأنواع حقول البيانات التي يوفرها الـ Table component للـ Data Source. ما زالت مسئوليتك أن تؤدي الحسابات الفعلية، أو أن تدخل قيم في أى حقول فعلية تضيفها. إن إضافة حقل باستخدام محرر الـ Fields لا يؤدي إلى إنشاء عمود جديد في جدول مجموعة البيانات- استخدم الـ Database Desktop لعمل التغييرات للمعلومات المادية الخاصة بالـ Table. يقوم محرر الـ Fields بإعادة تشكيل هذه المعلومات للاستخدام في التطبيق.

على سبيل المثال، بإعطاء مجموعة بيانات لها حقلى Date، فقد تريد أن تعرض عدد الأيام الواقعة بينهما. إن تخزين هذه المعلومة في مجموعة بيانات لا هدف منها لأنها قد تتغير كل يوم لكل تسجيل. هذا هو مثال نموذجي على نوع الحقل الذى يجب أن يحسبه التطبيق. إتبع هذه الخطوات:

١- أعد إضافة الـ DBGrid، إذا لزم الأمر، فى التطبيق وحدد خاصية الـ DataSource له بـ DataSource1. حدد خاصية الـ Active للـ Table1 بـ True. لعرض مجموعة البيانات الـ Grid. أعد التعليمات فى الفصل السابق إذا واجهت مشكلة، أو إذا لم تحفظ المشروع.

٢- اضغط مرتين يميناً أثناء الإشارة إلى نافذة المحرر، وأختر أمر الـ Add Field. اختر حقول الـ Name، الـ Source، الـ Vintage، والـ Purchased. (يمكنك استخدام مجموعة بيانات لها على الأقل حقل تاريخ واحد على الأقل). إرجع إلى شكل ١٧-١٦. اضغط OK لإغلاق هذه النافذة.

**Tip** فكرة: إذا أضفت حقل بطريق الخطأ، اضغط يميناً لظهور القائمة الخاصة بالمحرر واختر Delete لحذف الحقل. هذا يحذف فقط الـ object الذى يمثل الحقل. لا تتغير مجموعة البيانات الحقيقية الموجودة فى الملف.

٣- اضغط يميناً مشيراً داخل محرر الـ Field واختر أمر الـ New Field (راجع شكل ١٧-١٧). ادخل Name مثل Days Old للحقل الجديد، والذى يجب أن يختلف عن أسماء الحقول الأخرى. كما هو واضح فى شكل ١٧-١٧،

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

فإنك ترى اسم object (Table1 Days Old) في نافذة التحرير ثانية . هذا هو اسم ال object كما هو مقرر في ال form class - يمكنك تعديل الاسم الموّلد بصورة تلقائية في هذا الوقت ، أو يمكنك استخدام ال Object Inspector فيما بعد لتغيير خاصية ال object Name .

٤- اختر نوع Field لهذا الحقل الفعلي الجديد . في هذا المثال ، يكون ال IntegerField ملائماً . وهذا الخيار يحدد نوع البيانات ، مثل ال TStringField وال TIntegerField ، لتعريف ال object في ال form class . (كل حقل يصبح object في ال form) .

٥- تأكد من أن ال Calculated radio button قد تم اختياره ، ثم اضغط OK لإضافة حقل جديد لل Table1 . يمكنك عندئذ أن تغلق محرر ال Field أو تبعد نافذته جانباً . لاحظ أن ال DBGrid يعرض الآن عموداً جديداً يحمل labeled Days Old . (ال tool الرأسى إلى اليمين من عمود ال Name لتقصر هذا الحقل وتجعل الأعمدة الأخرى مرئية إذا لزم الأمر) . اختر أيضاً تعريف ال form class ، والذي يتضمن الآن تعريف ال object للحقل الفعلي الجديد :

Table1DaysOld: TIntegerField;

٦- إن توفير برمجة تحسب قيمة الحقل الجديد أمر يرجع إليك . افعل هذا باختيار ال Table1 Days Old باستخدام قائمة لائحة ال Object Inspector . (لا يمكنك اختيار هذا ال component في ال form لأن حقول مجموعة البيانات ليس لها تمثيل بصري) . اختر ال Object Inspector Events page ، ثم اضغط مرتين ال OnGetText لإنشاء event handler خالى . أدخل ال code التالية في ال procedure (ال procedure بأكمله مذكور هنا ك reference - إنك تحتاج أن تدخل فقط العبارة الوحيدة الواقعة بين ال begin وال end :

```
{ Calculate the Days Old virtual field using today's  
date (returned by the SysUtils Date function) and
```

```
the value of the database table's Purchased value. }
```

```
procedure TMainForm.Table1DaysOldGetText(Sender: TField;
```

```
var Text: String; DisplayText: Boolean);
```

```
begin
  Text := FloatToStr(Date - Table1Purchased.Value);
end;
```

٧- اضغط F9 لتشغيل البرنامج وعرض الحقول المحسوبة. يوجد التطبيق WinesCalc التام على القرص المدمج فى دليل Source\WinesCals.

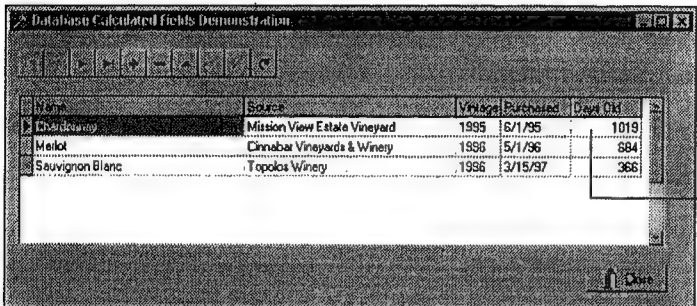
إن الـ Data-aware controls تطلق الـ OnGetText event للحصول على معلومات حقل. فى الـ event handler فى الـ code السابقة، عيّن الـ Text parameter البيانات التى تريد عرضها. فى هذا المثال، يحسب البرنامج عدد الأيام الواقعة بين اليوم وحقل تاريخ الـ Purchased. إن الـ Date function من الـ SysUtils unit تدخل تاريخ اليوم على أنه قيمة floating-point. وحقل الـ Purchased مذكور من خلال خاصية الـ Value للحقل الـ Table1Purchased. هذا هو واحداً من الـ objects التى أضفتها للـ Table باستخدام أمر الـ Add Field التابع لمحرر الـ Field.

ويعتمد نوع خاصية الـ Value للحقل على الـ object- فى هذه الحالة، للـ TDateField، تعتبر الـ Value هى نوع بيانات الـ floating-point لحساب عدد الأيام المنقضية بين اليوم وقيمة الحقل، يقوم البرنامج بطرح قيمتها ويمرر الناتج إلى الـ FloatToStr function التابعة لـ Delphi. يتم تعيين هذه النتيجة الـ Text parameter المتغير، والذى يتم استخدامه لعرض القيمة المحسوبة فى الـ DBGrid. يوضح شكل ١٧-١٨ البرنامج النهائى مع عمود الـ Days Old المحسوب فى أقصى يمين الـ Grid قاعدة البيانات.

**فكرة:** إن الحقول المحسوبة التى تولد exception قد تنشئ Loop لا نهائية من رسائل الخطأ. إذا ظننت أنك لن تستطيع التخلص من الأخطاء- وهذا الأمر شائع لإغلاق dialog الخطأ يؤدي إلى إعادة عرض النافذة، والتى تعيد حساب الحقل الخطأ، والذى يوكّد exception آخر- إرجع إلى Delphi، اختر الـ Tools\Environment Options... وفى صفحة الـ Debugger للـ dialog الناتج، اختر Delphi Exceptions واختر الـ User Program تحت عنوان Handled By. (فى Delphi 3 والنسخ السابقة له، كان

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

الخيار المساو لهذا هو مربع Break on Exception check box والذي يتم اختياره بواسطة أمر ال (Environment/Tools Options). ولأنك تقوم بتشغيل البرنامج في Delphi debugger، فإن تحديد هذا الخيار يعمل حتى مع التطبيقات التي يتم تنفيذها. هذا يجب أن يؤدي إلى إيقاف الأخطاء اللانهاية ويعيدك إلى Delphi.



Name	Source	Vintage	Purchased	Days Old
Chardonnay	Mission View Estate Vineyard	1995	6/1/95	1019
Merlot	Cinnabar Vineyards & Winery	1996	5/1/96	884
Sauvignon Blanc	Topolos Winery	1996	3/15/97	366

Calculated fields

شكل ١٧-١٨، تطبيق ال WinesCalc النهائي مع عمود ال Days Old المحسوب.

ويجب عليك أيضاً أن تختار ال (Ctrl+F2) Program Reset في قائمة ال Run. لا تقم بـ reboot حتى تجرب خطوات الإصلاح هذه. ولكن، في النهاية، قد يكون عليك أن تعيد ال reboot، ولكن إنهاء ال process يجب أن يصلح أى ذاكرة. إذا كنت تستخدم Windows 95 أو Windows 98. إنك غالباً لن تحتاج إلى إعادة ال reboot مع ال Windows NT، والذي يوفر تحكماً أعلى في الذاكرة. لاحظ أن هذه التقنية تعمل فقط إذا كنت تقوم بتشغيل تطبيق من داخل Delphi.

### ال Blob،

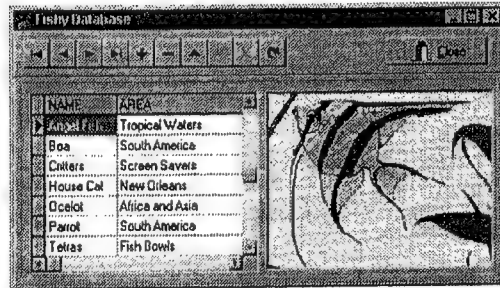
إن واحداً من أفلام الرعب القديمة المفضلة لدى (The Blob)، وهو يصور وحش بالابيض والاسود لزجة تحوّل ضحاياه إلى بقع على الأرض. وبالنظر إلى تكلفة هذه العملية السينمائية نرى أنها لا تتكلف أكثر من ثمن علبة الجيلاتين وعلبة سائل خفيف. إننى أعرف هذا لأن هذه هى المواد التى كنا نستخدمها لنصنع البقع الخاصة بنا فى الطرقات.

لحسن الحظ، إن حقول ال Blob الخاصة بـ Delphi ليست بهذه الدرجة من الرعب ولا هى لزجة مثلها. فى حالة أن تكون قد نسيت، فهذه ال Blob ما هى إلا



إختصار للـ Binary Large Object. في Table قاعدة بيانات، يستطيع حقل الـ Blob أن يخزن أى نوع من البيانات، ولكنه يستخدم بشكل نموذجي مع الصور الجرافيكية. على سبيل المثال، إن قاعدة بيانات خاصة بأصول عقارات قد تستخدم حقول الـ Blob لتخزين صور عن الأغراض المعروضة للبيع.

**على القرص المدمج:** إن تطبيق الـ Fishy على القرص المدمج، في دليل الـ Source\Fishy، يوضح كيفية استخدام مجالات الـ Blob. يوضح شكل ١٧-١٩ عرض تطبيق الـ Fishy. توضح القائمة ١٧-١ إلى source code للبرنامج. لتشغيل البرنامج، يجب أن تكون قد قمت بتركيب ملفات عرض Delphi، وخاصة أولئك على مسار الـ Demos\Data. يجب أن يكون هناك أسماء DBDEMOS مستعارة مسجلة لقاعدة البيانات هذه. إذا لم تستطع أن تجعل تطبيق الـ Fishy يعمل، استخدم الـ Database Desktop كما هو موضح في هذا الباب لتحديد موضع أسماء الـ DBDEMOS Alias إذا لم تجد هذه الأسماء، أعد تركيب Delphi.



شكل ١٧-١٩، يوضح تطبيق الـ Fishy قدرة Delphi على قراءة وكتابة حقول Blob في قواعد البيانات. في هذه الحالة، تكون الـ bitmap Blob موضحة إلى اليمين في نافذة البرنامج.

القائمة ١٧-١: Fishy\Main.pas.

```
unit Main;
```

```
interface
```

```
uses
```

```
SysUtils, Windows, Messages, Classes, Graphics,
```

Controls,  
Dialogs, StdCtrls, Forms, DBCtrls, DB, DBGrids,  
DBTables,  
Grids, Buttons, ExtCtrls;

type

```
TMainForm = class(TForm)
  ScrollBox: TScrollBox;
  DBNavigator: TDBNavigator;
  Panel1: TPanel;
  DataSource1: TDataSource;
  Panel2: TPanel;
  Table1: TTable;
  DBImage1: TDBImage;
  DBGrid1: TDBGrid;
  Table1NAME: TStringField;
  Table1AREA: TStringField;
  Table1BMP: TBlobField;
  BitBtn1: TBitBtn;
  procedure FormCreate(Sender: TObject);
  procedure DBImage1DblClick(Sender:
TObject);
private
  { private declarations }
public
  { public declarations }
end;
```

var

```
MainForm: TMainForm;
```

implementation

```
{ $R *.DFM }
```

```
{ - Open the Table1 dataset }
```

```

procedure TMainForm.FormCreate(Sender: TObject);
begin
  Table1.Open;
end;

{- Respond to double-click in Blob image }
procedure TMainForm.DBImage1DbClick(Sender: TObject);
begin
  with DBImage1.Picture do
    ShowMessage('W=' + IntToStr(Width) +
      ' H=' + IntToStr(Height));
  end;
end.

```

لا يوجد الكثير في قائمة تطبيق الـ Fishy - فإن أغلب عناصر البرمجة ثم السيطرة عليها من خلال objects الخاصة به. لإنشاء البرنامج، قمت باستخدام الـ Database Form Wizard، وفتحت أسماء الـ DBDEMOS Alias، واخترت ملف الـ Animals.dbf. بعد أن قام الـ Wizard بإنشاء الـ form، قمت بإزاحة الـ components جانباً و أضفت DBImage لعرض حقل الـ bitmap Blob، والذي يحمل BMP labeled في جدول قاعدة بيانات الـ Animals. فكرة: لكي تجرب نفسك في إنشاء نسخة خاصة بك من هذا البرنامج، حدد خاصية الـ DataSource للـ DBImage بـ DataSource1، وحدد الـ DataField بـ BMP. الباقي معروف وسهل.

لتوضيح كيف يمكنك برمجة events لمعلومات قاعدة البيانات، يقوم برنامج Fishy بتنفيذ الـ DBImage1 OnDbClick event. فلقد استخدمت الـ code الموضحة في آخر القائمة لتحديد حجم كبير نافذة الـ Blob النهائية (٢٠٠×١٦٠ في هذه الحالة). قم بتشغيل البرنامج، ثم اضغط مرتين صورة Blob لترى هذه القيم. في تطبيق أكثر توضيحاً، يمكنك تنفيذ نفس هذا الـ event لتحميل الـ bitmap، أو لتشغيل محرر تشغيل ثنائي لإنشاء وتعديل الـ bitmap المخزنة في قواعد البيانات. استخدم الـ DBImage1.Picture كما هو موضح في القائمة للوصول إلى الـ bitmap.

## الـ Structured Query Language

لقد ظهرت لأول مرة عام ١٩٨٦ بواسطة معهد المعايير الوطني الأمريكي (ANSI). ولأنه ليس لها هيكل تحكم، فإن الـ SQL لا يُعتقد أنها لغة برمجة. ولكن، تقوم الـ SQL بتعريف أوامر شبيهة بأوامر البرمجة مثل SELECT، JOIN، و UPDATE لأداء عمليات على Tables قاعدة البيانات. وتعتبر الـ SQL قديمة وصعبة الاستخدام، ولكن كل قواعد البيانات تقريباً تقدم على الأقل فرعاً من اللغة، لذا يمكنك الاستفادة من معرفة كيفية استخدامها.

**ملحوظة:** البعض ينطقون SQL على أنها "سكويل" والآخرين يقولون (إس، كيو، إل). والكثيرون يقولون مصطلحات أخرى لا مجال لذكرها هنا.

Note

لأن Delphi والـ BDE يؤديان أغلب خدمات قاعدة البيانات، إلا إذا كنت تطور تطبيقات الـ Server والعمل لقواعد بيانات الـ Remote SQL، فإنك غالباً تحتاج أن تتعلم أمر SQL واحد وهو: SELECT. كما سأوضح في هذا الفصل، يمكنك استخدام هذا الأمر لأداء أنواع كثيرة مختلفة من البحوث على المعلومات المخزنة في جداول قواعد البيانات.

إن الـ BDE يؤيد فرعاً من الـ SQL المعيارية للـ dBase، Paradox، Oracle، وقواعد بيانات أخرى. إذا كانت tables قواعد بياناتك ضمن الـ form المؤيدة، فيمكنك استخدام أوامر الـ SQL الذاتية الوضوح SELECT، INSERT، UPDATE، و DELETE لأداء هذه العمليات. ولبعض الـ SQL server الآخرين يوفرون أوامر إضافية- إرجع إلى وثيقة المساعد الخاصة بك لمعرفة الأوامر التي يمكنك استخدامها بالضبط.

### الـ Query

تستخدم تطبيقات الـ Database تطبيقات الـ Query لإصدار عبارات SQL إلى قاعدة بيانات. إنك تحتاج Query object واحد لكل قاعدة بيانات ترسل إليها بأوامر الـ SQL. في تطبيقات Delphi، إنك غالباً ما تستخدم Query objects لأداء بحوث تعتمد على قضايا مثل "كل الأشخاص الذين لهم أعين بنية والذين ولدوا يوم الثلاثاء ويتنمون إلى رابطة البولنج" والمعايير الغريبة الأخرى.

إن حالة من ال Query component تشبه ال Table لأن ال Query يوفر منفذاً إلى قاعدة البيانات من خلال اسم alias مسجل . بالإضافة إلى ال Query ، فإنك تحتاج إلى ال DataSource لربط ال Query بال data-aware controls . وتوضح ال controls بشكل نموذجي النتائج أوامر ال SQL المنفذة .

## إنشاء محرر SQL:

**على القرص المدمج:** إن تطبيق ال SQLPlay على القرص المدمج ، في دليل ال Source\SQLPlay ، يوفر مكاناً للبدء في إنشاء محرر لإصدار أوامر ال SQL إلى قاعدة بيانات مختارة . هذا يوضح كيفية إصدار والاستجابة لأوامر ال SQL باستخدام ال Query . يوضح البرنامج أيضاً كيفية استجواب ال BDE عن كل أسماء ال Tables وأسماء Alias لقاعدة البيانات - إنك تريد أن تفعل هذا في تطبيقك لتوفر سبلاً للمستخدمين لاختيار جداول قاعدة البيانات المتاحة .



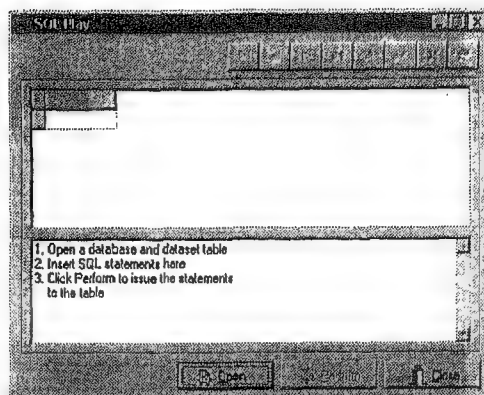
قم بتشغيل تطبيق ال SQLPlay الآن بتحميل SQLPlay.dpr في Delphi وضغط F9 . يوضح شكل ١٧-٢٠ النافذة الرئيسية للبرنامج قبل فتح أى جداول قاعدة بيانات .

اضغط زر ال Open لل SQLPlay لعرض ال dialog box الموضح في الشكل ١٧-٢١ . استخدم هذا ال dialog لاختيار قاعدة بيانات وجدولاً مثل DBDEMOS و CUSTOMER ، ثم اضغط OK لتعود إلى النافذة الرئيسية للبرنامج .

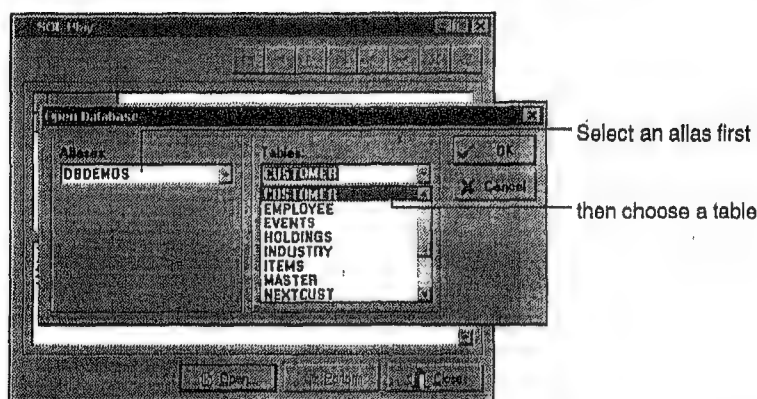
قم بإدخال أوامر ال SQL في Memo تحت ال DBGrid الذى يوضح معلومات قاعدة البيانات . اضغط Perform لتنفيذ أوامرك . على سبيل المثال ، افتح قاعدة بيانات DBDEMOS وجدول Customer . ثم قم بتحرير أمر نافذة ال Memo إلى السطور التالية ، والتي تبحث عن كل العملاء في كاليفورنيا . اضغط Perform أو اضغط Alt+P لإصدار أمر ال SQL لجدول قاعدة البيانات :

Select \* From CUSTOMER

where State="CA"



شكل ٢٠-١٧: هذا هو عرض برنامج الـ SQLPlay قبل فتح جدول قاعدة بيانات.



شكل ٢١-١٧: الـ Open dialog للـ SQLPlay يذكر قائمة بالـ Tables وأسماء الـ Alias لقاعدة البيانات المتاحة.

توضح القوائم ١٧-٢ الـ SQLPlay dialog module، وهي Open.pas. وهذه الـ unit توضح كيفية الحصول على كل أسماء الـ Alias المسجلة لقواعد البيانات وأسماء Tables مجموعة البيانات الخاصة بها وكيفية تقديم هذه المعلومات في قائمة لائحة يستطيع المستخدمون أن يختاروا منها Table مجموعة بيانات لتحريره ورؤيته.

القائمة ٢-١٧: SQLPlay\Open.pas

```
unit Open;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics,  
Controls,  
Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, Db,  
DbTables;
```

```
type
```

```
TOpenForm = class(TForm)  
    OKBtn: TBitBtn;  
    CancelBtn: TBitBtn;  
    Bevel1: TBevel;  
    ComboBox1: TComboBox;  
    Label1: TLabel;  
    Bevel2: TBevel;  
    Label2: TLabel;  
    ComboBox2: TComboBox;  
    procedure FormActivate(Sender: TObject);  
    procedure ComboBox1Change(Sender: TObject);  
private  
    { Private declarations }  
public  
    { Public declarations }  
end;
```

```
var
```

```
OpenForm: TOpenForm;
```

```
implementation
```

```
{ $R *.DFM }
```

```
procedure TOpenForm.FormActivate(Sender: TObject);  
begin  
    Session.GetAliasNames(ComboBox1.Items);
```

end;

```
procedure TOpenForm.ComboBox1Change(Sender: TObject);
```

```
begin
```

```
  Session.GetTableNames(ComboBox1.Text, '.*',
```

```
    False, False, ComboBox2.Items);
```

```
  ComboBox2.ItemIndex := 0;
```

```
end;
```

```
end.
```

عند تنشيط ال form (انظر ال procedure FormActivate، للـ OnActivate event الخاص بهذه ال form)، تقوم ال dialog box unit باستدعاء ال GetAliasNames للحصول على قائمة string بأسماء ال Alias لقاعدة البيانات من ال BDE. وهذا ال method - كثيرين غيره - مزود بـ TSession class لاستخدام هذا ال object، أضف Db إلى عبارة ال uses الخاصة بال moudle، واستدع ال TSession methods بالإشارة إلى ال Session، والذي ينشئه Delphi بصورة تلقائية لكل تطبيقات قواعد البيانات. إن ال GetAliasNames يسمح ويدخل أسماء Alias في خاصية ال TStrings أى ال TStringList. هنا، يستخدم البرنامج ال method لبدء بنود ال ComboBox1.

لأن أسماء ال Table تعتمد على أى أسماء ال Alias يختار المستخدم، فإن ال ComboBox الثانى لا يمكن بدئه فى ال FormActivate. بدلاً من ذلك، فبعد أن يختار المستخدم قاعدة بيانات، فإن ال ComboBox1Change procedure - والذي يتم اطلاقه عندما تتغير نافذة تحرير ال control - يستدعى TSession method آخر، وهو ال GetTableNames، للملء معلومات ال ComboBox2. يقوم ال TSession بتعريف ال GetTableNames وال parameters الخمسة التابعة لها كما يلى:

```
procedure GetTableNames(const DatabaseName, Pattern: string;
```

```
  Extensions, SystemTables: Boolean; List: Tstrings);
```

● **DatabaseName** - هذا هو اسم قاعدة بيانات أو Alias name مثل

.DBDEMOS



● **Pattern** - حدد هذا بامتداد اسم الملف مثل \*.db لتقصر اسماء ال Table على الملفات المناسبة .

● **Extensions** - حددها ب True إذا كنت تريد للأسماء المدخلة أن تتضمن إمتدادات اسم ملف . يكون هذا ال parameter ذا قيمة لتطبيقات قاعدة البيانات للحاسب المكتبي فقط .

● **SystemTables** - حدده ب True للحصول على System Table وكذلك لكي تستخدم مجموعات البيانات من remote server . يكون هذا ال parameter ذا قيمة لتطبيقات قاعدة بيانات ال Server والعميل فقط .

● **List** - قم بتمرير أى خاصية TStrings أو إلى هذا ال parameter . يقوم ال GetTableNames بمسح القائمة وإدخال اسماء Table مجموعة بيانات فى القائمة . يستخدم التطبيق هذا ال parameter لبدء خاصية Items لل ComboBox2 ، وهو object كل من نوع ال TStrings . إن تحديد ال Item Index التابع لهذا ال object بصفر يعرض المدخل المذكور أولاً فى نافذة تحرير صندوق ال Combo .

توضح القائمة ١٧-٣ التركيبة الرئيسية لل SQL Play ، والتي تؤدي أوامر ال SQL وتعرض tables قاعدة البيانات المختارة . بعد القائمة ، سوف أوضح كيف يمكن لل form Objects الرئيسية أن تتعاون لتعطى نتائج بحث قاعدة البيانات فى ال DBGrid (يمكنك أن تستخدم نفس ال methods مع أى data-aware controls .

القائمة ١٧-٣: SQLPlay\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
SysUtils, Windows, Messages, Classes, Graphics,  
Controls,
```

```
Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, Grids,
```

DBGrids, DBCtrls, DB, DBTables, Open;

type

```
TMainForm = class(TForm)
  DataSource1: TDataSource;
  DBNavigator1: TDBNavigator;
  DBGrid1: TDBGrid;
  Memo1: TMemo;
  Bevel1: TBevel;
  PerformBitBtn: TBitBtn;
  CloseBitBtn: TBitBtn;
  OpenBitBtn: TBitBtn;
  Query1: TQuery;
  procedure OpenBitBtnClick(Sender: TObject);
  procedure PerformBitBtnClick(Sender: TObject);
  procedure FormClose(Sender: TObject;
    var Action: TCloseAction);
private
  { Private declarations }
public
  { public declarations }
end;
```

var

MainForm: TMainForm;

implementation

{ \$R \*.DFM }

```
procedure TMainForm.OpenBitBtnClick(Sender: TObject);
begin
  if OpenForm.ShowModal = mrOk then
    begin
      Query1.Close;
```

```

try
    Query1.DatabaseName := OpenForm.ComboBox1.Text;
    Query1.SQL.Clear;
    Query1.SQL.Add('Select * From ' +
    OpenForm.ComboBox2.Text);
    Memo1.Lines := Query1.SQL;
    Query1.Open;
    Memo1.SetFocus;
    PerformBitBtn.Enabled := True;
except;
    ShowMessage('Unable to open database');
end;

end;
end;

procedure TMainForm.PerformBitBtnClick(Sender: TObject);
begin
    Query1.Close;
    try
        Query1.SQL := Memo1.Lines;
        Query1.Open;
    except
        ShowMessage('Invalid query');
    end;
end;

procedure TMainForm.FormClose(Sender: TObject;
    var Action: TCloseAction);
begin
    Query1.Close;
end;

end.

```

في قائمة البرنامج الرئيسى، يعرض الـ `OpenBitBtnClick` procedure نافذة `OpenForm` باستدعاء الـ `ShowModal`. إذا أنهى المستخدم الـ `dialog`

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

بضغط OK، يغلق البرنامج ال Query1 للإعداد لفتح قاعدة بيانات جديدة. تعمل ال Query objects مثل ال Tables، ولكنها تستطيع أيضاً أن تنفذ عبارات SQL. وهذا يستلزم بعض عبارات الاعداد. أولاً، يقوم البرنامج بتعيين DatabaseName من نافذة تحرير ال ComboBox الخاصة بال dialog. ثم نحن بحاجة إلى عبارتين لمسح قائمة ال SQL string الحالية في ال Query، واستدعاء ال Add لادخال عبارة مثل هذه:

Select \* From Animals

إن أمر ال SQL Select يبحث في مجموعة بيانات مثل Animals عن مدخلات متشابهة. وتختار علامة النجمة كل أعمدة (أو حقول) ال Animals table. ولأنه لم يتم استخدام عبارة WHERE لتحديد من اختيار الصفوف، فإن كل الصفوف (أو السجلات) في ال Animals table قد تم ضمها إلى المجموعة الناتجة. ولكي يمكنك من تحديد أوامر ال SQL، يقوم البرنامج بتعيين خاصية ال SQL التابعة لل Query1 إلى Memo. ثم يستدعي البرنامج ال Open for ال Query1، والذي يصدر عبارة SQL إلى مجموعة البيانات.

**ملحوظة:** إن استدعاء ال Open لل Query يصدر أمر ال SQL Select فقط. ولأوامر ال SQL الأخرى مثل Insert و Delete، استدع ال ExecSQL method.

Note

إن ال PerformBitBtnClick procedure ينفذ أوامر ال SQL التي تدخلها في نافذة Memo. أولاً، يغلق البرنامج ال Query1. (أغلق ال Queries وال Tables دائماً قبل عمل أى تغييرات هامة في خصائصهما). قم بتعيين TStrings أو TStringList إلى خاصية ال SQL، ثم افتح قاعدة البيانات لإصدار الأمر، والذي يكون في هذا المثال Select.

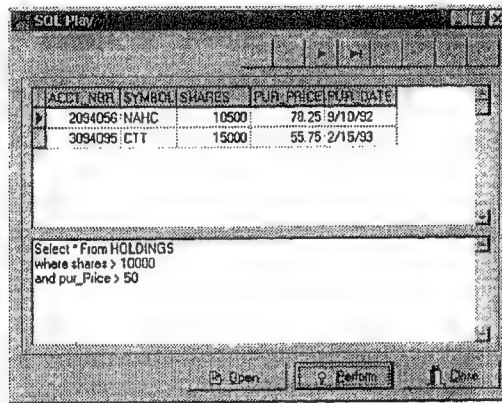
وللتجربة ببرنامج ال SQLPlay، قم بتشغيل البرنامج وافتح قاعدة بيانات ال DBDEMOS. اختر table مجموعة بيانات ال HOLDINGS. اضغط OK لتعود إلى نافذة البرنامج الرئيسية، واضغط داخل ال Memo pad. قم بتحرير عبارة ال SQL الأصلية إلى الآتى، واضغط Alt+P لأداء الأمر، والذي يبحث في مجموعة البيانات عن كل الممتلكات تكون فيها الأسهم أكبر من ١٠٠٠٠:

Select \* From HOLDINGS

where shares >10000

يمكنك الاستمرار في تنقية البحث . على سبيل المثال ، أضف تحديداً آخر ، كما يلي ، تحت السطرين السابقين لعرض الأسهم التي تزيد عن ١٠٠٠٠ ولها سعر شراء أكثر من ٥٠ دولار . يوضح شكل ١٧-٢٢ النافذة الناتجة . (إن تطبيق الـ SQLPlay على القرص المدمج يوضح كيفية إصدار والاستجابة إلى أوامر الـ SQL).

and pur\_Price > 50



شكل ١٧-٢٢: هنا، يتم توضيح بحث في الـ Holdings DBDEMOS Table للسجلات ذات الاسهم التي تزيد عن ١٠٠٠٠، ولها سعر صرف أعلى من ٥٠ دولار.

**ملحوظة:** بعد فتح قاعدة بيانات بواسطة الـ SQLPlay Open Button ، يمكنك إختيار database Table آخر دون إعادة فتح الـ dialog . على سبيل المثال ، إستبدل كل السطور في نافذة الـ Memo بـ Select \* From Animals ، واضغط الـ Alt+P لفتح الـ Animals table وعرض كل سجلاته .



## قواعد بيانات Master-Detail

إن أغلب قواعد البيانات علاقية ، وهذا يعنى ببساطة إنها لها إثنين أو أكثر من الـ Tables ترتبط ببعضها بواسطة حقول عامة . على سبيل المثال ، إن الـ Customer table قد يكون له حقل ID يستخدم أيضاً في الـ Purchases tables لتعريف كل شراء يكون قد قام به عميل بعينه . وهذا الإعداد يسمى علاقة

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

(one-to-many). يتم تعريف السجلات المنفردة بطريقة فريدة بواسطة مفتاح (حقل الـ ID) في table الأصل (Customer). ويكون واحداً أو أكثر من السجلات مرتبط بالعملاء في جدول التفاصيل (Purchases). وهذه العلاقة تسمى (master-detail model).

### فهم (master-detail model)

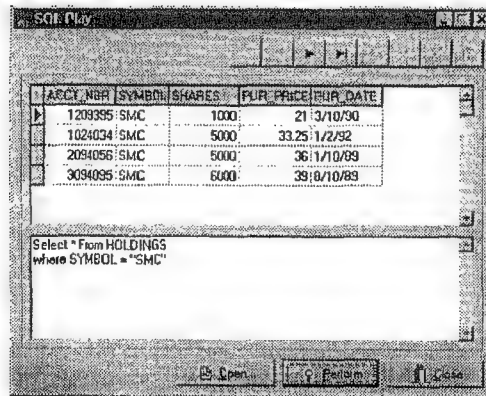
يأتي Delphi بالعديد من الأمثلة المفيدة من tables قاعدة بيانات (master-detail). وكمثال، قم بتشغيل تطبيق الـ SQLPlay من الفصل السابق وأفتح الـ Alias name المستعار DBDEMOS اختر الـ MASTER table واضغط OK لتعود إلى النافذة الرئيسية. يذكر الـ DBGrid object اسهم متنوعة، كل منها معرف بطريقة فريدة بواسطة primary key يسمى SYMBOL.

الآن، أدخل عبارة الـ SQL هذه في نافذة الـ Memo:

```
Select * From HOLDINGS
```

```
where SYMBOL = "SMC"
```

اضغط Alt+P لأداء العبارة وأختر كل الـ HOLDINGS ذات الرمز SMC. وtable الأصل له مدخل واحد فقط مثل هذا؛ وجدول التفاصيل يمكن أن يكون له سجلات عديدة، واحداً لكل شراء لهذا السهم بالذات. يوضح شكل (١٧-٢٣) نافذة الـ SQLPlay بعد أداء هذا البحث.



شكل (١٧-٢٣)، الـ SQLPlay بعد البحث في  
الـ HOLDINGS على حقل الـ SYMBOL العام

## برمجة تطبيقات (master-detail)

إن خصائص الـ MasterSource والـ MasterFields للـ Table تعرف علاقات الـ master-detail فى تطبيقات قاعدة بيانات Delphi. أستخدم هذه الخصائص كما يلى:

● MasterSource - وكما أيضاً detail Table object، عين اسم DataSource object مرتبط بـ master Table. أترك هذه الخاصية خالية فى الـ master Table.

● MasterFields - كذلك للـ detail Table عين string له واحداً أو أكثر من أسماء الحقل إلى هذه الخاصية، التى تعرف الحقل (العمود) الذى يرتبط فوقه اثنين من الـ Tables.

يمكنك تحديد حقول متعددة منفصلة عن بعضها بفصلة منقوطة فى string MasterFields. ها هى بعض العينات التى توضح الـ formats الممكنة:

DetailTable.MasterFields := 'Symbol';

DetailTable.MasterFields := 'Symbol;Exchange';

وبخلاف بعض الآراء، لا يجب عليك أن تحدد MasterFields فى Table. قم بتعيين string لهذه الخاصية فقط فى الـ detail Table object. بالإضافة إلى ذلك، حدد خصائص الـ IndexFieldNames أو الـ IndexName التابعة لـ detail Table. إستخدم دائماً الـ IndexName متى أمكن لأنها تساعد فى عملية lookups. حدد الـ IndexFieldNames عندما لا يوجد فهرس للـ joined field.

**على القرص المدمج:** كمثال على كيفية إنشاء تطبيق قاعدة بيانات الـ master والـ detail قم بتشغيل برنامج الـ Master على القرص المدمج فى دليل الـ Source\Master. يستخدم هذا البرنامج الـ Master

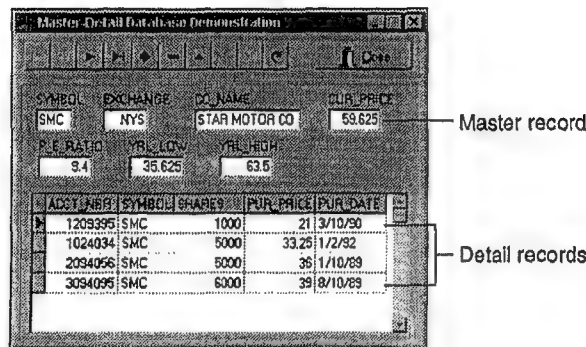


tables والـ Holdings من قاعدة بيانات الـ DBDEMOS المتوفرة مع Delphi. يوضح شكل ١٧-٢٤ عرض البرنامج. إضغط أزرار الـ DBNavigator لتصفح الـ Master Table وعرض المدخلات من Holdings table مع حقول Symbol الملائمة. توضح الـ code التالية البرمجة الوحيدة المهمة فى هذا التطبيق: الـ

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

form OnCreate، الذى يفتح اثنين من ال Tables لقاعدة البيانات. وتعتنى ال Components وخصائصها بكل الجوانب الأخرى لعملية هذا البرنامج - وهى مظهر قوى لقدرات البرمجة البصرية لـ Delphi.

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
    MasterTable.Open;
    DetailTable.Open;
end;
```



شكل (١٧-٢٤): اضغط أزرار ال DBNavigator لترى الأسهم من Master table وتوضح أية ممتلكات لتلك الأسهم فى ال DBGrid من ال فشر مث detail Holdings.

يوضح جدول ١٧-٢ قيم الخاصية الهامة لل DataSource وال Table التابعة لبرنامج ال Master. بشكل عام، فإن كل علامة master-detail تحتاج إلى زوجين من هذه ال objects المتعلقة بواسطة الخصائص الموضحة فى ال Table. ويظهر ال Table أيضاً data-aware control (EditSYMBOL) مرتبط بال MasterSource، ال DetailDBGrid مرتبط بال DetailSource. وهذه البنود تمثل الحد الأدنى من العناصر المطلوبة لتطبيق قاعدة بيانات ال master-detail. (بالمناسبة، لقد استخدمت ال Database Form Wizard لتبدأ هذا التطبيق، الذى قمت بتعديله بإعادة تسمية ال components، وبإدخال object DBGrid، وبعمل تعديلات بسيطة للعرض).



## إستخدام Data Modules:

ليس من غير الشائع للعديد من التطبيقات أن تتوصل إلى نفس database tables. على سبيل المثال، بمصاحبة قاعدة بيانات مطولة، فقد يطلب منك أن تنشئ عشرات من التطبيقات لبحث، وتسهيل، وتصنيف معلومات قاعدة البيانات. فى تلك الحالات، من المستحسن عادة أن تنشئ Data Module (أو Data Module) تعرف القواعد واللوائح للتوصل إلى database tables. بتركيب ال Data Module فى ال Object Repository الخاص بـ Delphi، فيمكنك أنت والمبرمجون الآخرون أن تبدأوا بسهولة تطبيق قاعدة بيانات جديد.

جدول (١٧-٢)، خصائص ال DataSource وال Table لتطبيق ال Master.

Component	Name	Property	Value
Table	MasterTable	DatabaseName	DBDEMOS
		TableName	master.dbf
DataSource	MasterSource	DataSet	MasterTable
Table	DetailTable	DatabaseName	DBDEMOS
		IndexName	SYMBOL
		MasterFields	SYMBOL
		MasterSource	MasterSource
		TableName	holdings.dbf
DataSource	DetailSource	DataSet	DetailTable
DBEdit	EditSymbol	DataField	SYMBOL
		DataSource	MasterSource
DBGrid	DetailDBGrid	DataSource	DetailSource

تحتوى ال Data Module النموزجية على ال Table وال Query وال DataSource وcomponents أخرى والتي تعرف بـ Alias name، اسم ال Table وخصائص أخرى. يمكن لأي عدد من التطبيقات أن يضيف ال Data Module ملفات المشروعات الخاصة بهم، ويبدأوا فى إستخدام Tables قاعدة البيانات على الفور. وهذا التنظيم أيضاً حافظ على components قاعدة البيانات الغير بصرية فى نافذة منفصلة غير مرئية، بدلاً من التكدس فى form وقت

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

التصميم . ولكن الميزة الحقيقية لإستخدام الـ Data Modules هي توفير تطبيقات متعددة مع المواد الهام اللازمة للتوصل إلى قاعدة البيانات . بالإضافة إلى ذلك ، إذا تغيرت القواعد واللوائح - كما هو الحال غالباً - يمكنك ببساطة تحديث الـ Data Modules وإعادة الـ compile لتطبيقاتك بدلاً من إعادة برمجة الـ DataSource والـ Table في كل برامجك .

**ملحوظة:** كمقدمة حول إستخدام الـ Data Modules والـ Object Repository ، أنظر الباب الثالث ، «معلومات حول الـ forms» ، تحت عنوان Form Templates .

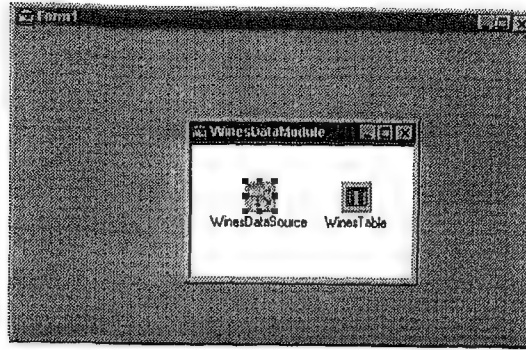
Note

إتبع هذه الخطوات لإنشاء Data Module لقاعدة بيانات الـ Wines في هذا الباب ، ولتركيبتها في الـ Object Repository الخاص بـ Delphi (الملفات الناتجة ليست على القرص المدمج - فيجب أن تنشئها لكي تستخدم المعلومات الواردة في هذا الفصل):

١- قم بتوفير دليلاً لتخزين الملفات . أبدأ تطبيقاً جديداً ، ثم إختتر أمر الـ File/New Data Module .

٢- إستخدم الـ Object Inspector ليصبح WinesDataModule . يمكنك حفظ كل الملفات عند هذه النقطة لإقامة الدليل المنشود . أعد تسمية الملف المرتبط بالـ Data Module - (وهو Unit2.pas في هذا المثال) ليصبح شيئاً مناسباً مثل WinesDM.pas .

٣- قم بإظهار نافذة الـ WinesDataModule مرة أخرى (إستخدام أمر الـ View/Window List إذا لم تجدها) . في هذه النافذة ، قم بإسقاط DataSource وTable من الـ Data Access . قم بتغيير خاصية الـ DataSource1 Name لتصبح WinesTableSource . قم بتغيير خاصية الـ Table 1 Name لتصبح WinesTable . يوضح شكل ١٧-٢٥ نافذة الـ Data Module في هذه المرحلة . تكون نافذة الـ Form1 خالية ولن تستخدم (ولكن ، يمكنك تطويرها لتكون برنامج إختياري للـ Data Module هذه) .



شكل (١٧-٢٥): نافذة الـ WinesDataModule ذات الـ DataSource والـ Table.

٤- بعد ذلك تقوم بتطوير القواعد الخاصة بالتوصل إلى قاعدة البيانات باستخدام الـ WinesTable، وفي الـ Object Inspector، إضغط السهم المشير إلى أسفل الواقع بعد خاصية الـ DatabaseName، وإختر WINES. بنفس الطريقة، إختر Wines.db لخاصية الـ TableName.

٥- إختر الـ WinesDataSource في الـ Data Module ثم، باستخدام الـ Object Inspector، حدد خاصية الـ DataSet بـ WinesTable.

٦- إحتفظ كل الملفات بإختيار File|Save All أو بضغط على Save All Button.

تم الخطوات السابقة البرمجة للـ Data Module. في تطبيقاتك، تكون حراً في إضافة components إضافية وتحديد خصائص أخرى للوصول إلى Tables قاعدة البيانات الخاصة بك. بالرغم من أن المثال تستخدم إثنين فقط من الـ components، فقد يكون الـ Data Module الأكثر تعقيداً يملك أكثر Table مختلفه و DataSource و component قاعدة بيانات أخرى عديدة للوصول إلى العديد من الـ Tables.

**فكرة:** لا تهتم بتحديد خاصية الـ Table Active بـ True. فإن هذه تتغير إلى False عندما تقوم بتخزين الـ Data Module في الـ Object Repository.

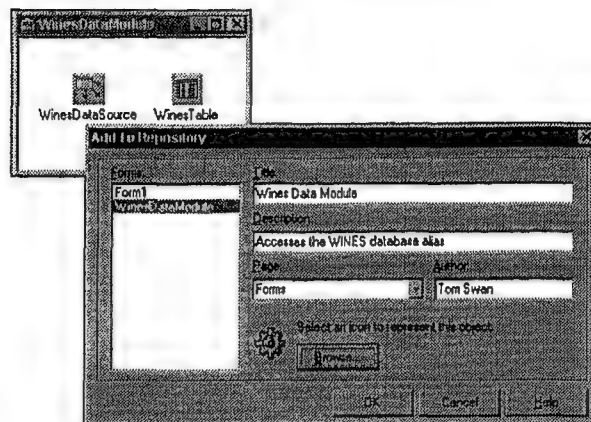


لكي تجعل الـ Data Module متاحة لتطبيقات أخرى، قم بإضافتها إلى الـ Object Repository الخاص بـ Delphi (سوف أشرح كيفية حذف المدخل، فلا

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

تتردد في أن تجرب هذا). أتبّع هذه الخطوات لإضافة الـ Data Module من الخطوات السابقة إلى الـ Object Repository.

١- إجعل نافذة الـ WinesDataModule مبرئية (إستخدم الـ View/Window List إذا لزم الأمر لتحديد موضع النافذة). إضغط يميناً أثناء الإشارة بمؤشر الفأرة داخل النافذة. هذا يؤدي إلى إظهار قائمة اختر أمر الـ ... Add to Repository. يوضح شكل ١٧-٢٦ الـ dialog الناتج.



شكل (١٧-٢٦)، بالإضافة إلى الـ Add to Repository dialog.

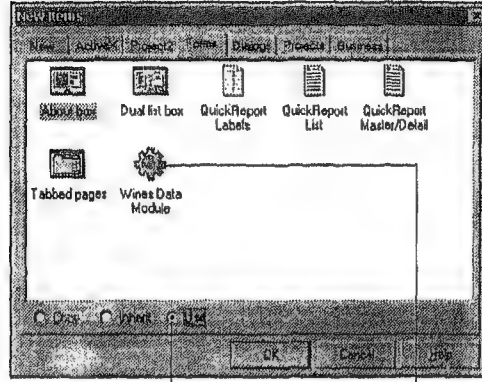
٢- اختر الـ WinesDataModule في قائمة الـ Forms (يجب أن يتم إختيارها حسب النظام الافتراضي). قم بإدخال Title و Description، وأختر Page حيث تريد أن تظهر الـ Data Module عندما تختار الـ File/New.... يمكنك أيضاً أن تملأ حقل الـ Author وتضغط Browse لإختيار ملف أيقونة لعرض صورة في الـ New Items dialog. (لقد أستخدمت أيقونة مقدمة مع Delphi في دليل الـ Images\Icons). يوضح شكل ١٧-٢٦ القيم التي ملأتها.

٣- إضغط Ok لإغلاق الـ Add to Repository dialog إذا لم تكن قد حفظت الـ module المضافة، فإنه سيطلب منك أن تفعل هذا قبل تضمينها في مخزن Delphi.

يمكنك الآن إستخدام الـ Data Module في أى تطبيق هذا أمر بسيط - فقط اخترها بإستخدام أمر الـ File/New. حاول تجربة هذه الخطوات لتختبر الـ Wines Data Module

١- إبدأ تطبيقاً جديداً.

٢- اختر File/New... واضغط Forms page tab في ال New Items dialog. يجب أن ترى ال Wines Data Module التي تم إضافتها حديثاً. (أنظر شكل ١٧-٢٧).



First choose the Use button and then double-click this icon

شكل (١٧-٢٧): اختر ال Wines Data Module التي تم إضافتها حديثاً من ال New Items dialog.

٣- اضغط الزر الذي يحمل بطاقة Use قرب أسفل نافذة ال dialog (راجع شكل ١٧-٢٧). أنظر الملاحظة الموجودة بعد هذه الخطوات للحصول على تفسير للطرق الثلاثة الممكنة لتضمين ال module : Copy ، Inherit ، و Use.

٤- اضغط مرتين أيقونة ال Wines Data Module. هذا يفتح نافذة ال Data Module في التطبيق الحالي وأضف source Pascal code الخاصة به إلى نافذة ال code.

٥- إنتقل إلى نافذة ال code، وأختر Unit1 (ال form source code الرئيسية للبرنامج). ضع تعريف ال uses في أعلى الشاشة وقم بتعديله كما يلي. هذا التغيير يجعل ال WinesDM ، Data Module (هذا هو الاسم الذي حفظت به ال source Pascal code لل module ، متاحة لل form unit الرئيسية. بالتبادل، يمكنك استخدام أمر ال File/Use Unit الخاص بـ Delphi لتقوم بهذا التغيير بصورة تلقائية، ولكن لقطاع ال implementation الخاص بال unit المضيفة بدلاً من واجهة التطبيق كما هو موضح هنا:

uses

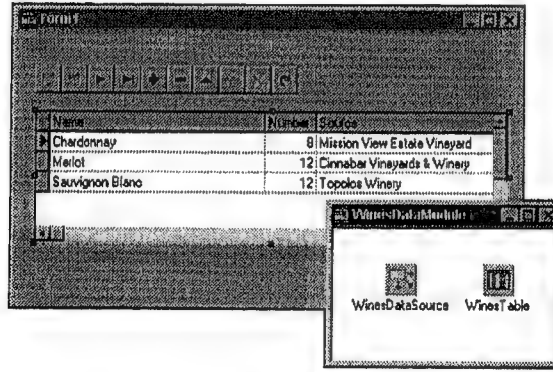
Windows, Messages, SysUtils, Classes, Graphics, Controls,

Forms, Dialogs, WinesDM;

٦- أضف DBNavigator وDBGrid من ال Data Controls palette إلى نافذة ال form الرئيسية. اختر كلا ال objects (clock+Shift) لكل object). في نافذة ال Object Inspector، حدد خاصية ال DataSource، حدد خاصية ال DataSource لكل ال object بما يلي (هذا هو الاختيار الوحيد في هذا المثال):

WinesDataModule.WinesDataSource

٧- إرجع إلى نافذة ال Wines Data Module إذا لزم الأمر). اختر ال WinesTable، وإستخدم ال Object Inspector لتحديد خاصية ال Active به True. هذا يفتح قاعدة بيانات ال Wines ويظهر البيانات الحية في نوافذ ال DBNavigator وDBGrid (أنظر شكل ١٧-٢٨).



شكل (١٧-٢٨): هذا البرنامج، الذي تم إنشاؤه بإتباع الخطوات في هذا الفصل، يستخدم ال Wines Data Module للوصول إلى قاعدة Wines Data Module للوصول إلى قاعدة بيانات ال Wines.

إن التطبيق الذي أنشأته لتوك بإستخدام ال Wines Data Module يوضح طريقة عظيمة للتشارك في objects الوصول إلى قاعدة البيانات مع تطبيقات متعددة إنك لا تحتاج إلى DataSource في ال form الرئيسية للبرنامج لأن هذه متوفرة في ال Data Module. تستطيع التطبيقات الأخرى أن تستخدم نفس ال

Data Module للتمكن من الوصول إلى قاعدة بيانات الـ Wines. إذا احتاجت هذه الـ objects إلى التغيير - مثلاً، لتغيير الـ alias name، اسم الـ Table، أو لإضافة حقول وهمية كما هو موضح في هذا الباب باستخدام الـ phantom Fields - فإنك ببساطة تقوم بإعادة الـ compile لتطبيقاتك لتضمين التغييرات.

**ملحوظة:** إن البنود الموجودة في الـ Object Repository يمكن إضافتها إلى مشروع باستخدام ثلاثة خيارات: Copy، Internet، و Use (لاحظ أن ليس كل الخيارات متاحة لكل نوع من البنود - المشروعات، مثلاً، يمكن نسخها فقط). عندما تقوم بنسخ عنصر Object Repository، يصنع Delphi نسخة كاملة للملفات ذلك العنصر في دليل مشروعاتك. أية تغييرات تفعلها بهذا العنصر توجد في هذه النسخة فقط. عندما تورث بنداً، ينشئ Delphi الـ class الجديدة على أساس item's class مع هذا الخيار، يمكنك فعل تغييرات، ولكن بواسطة إعادة الـ compile، قم بتوريث أية تغييرات صنعت بالعنصر في الـ Object Repository. عندما تستخدم بنداً، يشير Delphi إلى ملفاته أينما كانت مخزنة. أية تغييرات تحدث بالعنصر تنعكس في تطبيقك عندما يتم الـ compile. إن قواعد بيانات الـ Data Modules أفضل ما تكون عند توريثها أو استخدامها.

## برمجة قاعدة البيانات

تصف الفصول التالية بعض البنود المتنوعة حول برمجة قاعدة البيانات مع Delphi والتي قد تجدها معونة لك.

### خاصية الـ TDataSet CacheBlobs

إن الـ TDataSet class توفر الآن خاصية، وهي الـ CacheBlobs، يمكنك أن تحددها بـ True أو False إن الـ CacheBlobs يتم توارثها من جانب كل الـ class المنحدرة من جانب الـ class المنحدرة من الـ TDataSet، مثل الـ TStoredProc، TQuery، TTable، TBDEDataSet.

حدد الـ CacheBlobs بـ True لتخفظ في الذاكرة أي عدد من حقول الـ Blob بقدر الإمكان، حتى عندما تكون السجلات المرتبطة بها ليست في الذاكرة أي

## الباب السابع عشر : تطوير تطبيقات قاعدة البيانات

عدد من حقول الـ Blob بقدر الإمكان، حتى عندما تكون السجلات المرتبطة بها ليست في الذاكرة.

حدد الـ CacheBlobs بـ False لإخراج حقول الـ Blob من الذاكرة مع السجلات الأخرى لقاعدة البيانات.

والطبيعي هو أن تكون الـ CacheBlobs محددة بـ True. هذا يؤدي إلى تحسين الأداء خاصة في البرنامج التي تعرض Bitmap من حقول الـ Blob. عندما يرجع المستخدم إلى سجل ذا CacheBlobs، فيتم تحميله من الذاكرة بدلاً من Table قاعدة البيانات على قرص أو gride محددة بـ True.

### الـ TDataSet مقابل الـ TDataSet

في Delphi 4، لم تعد الـ TDataSet class معتمدة على الـ BDE. وهذا التغيير ليس له أثر على التطبيقات الموجودة، ولا تؤثر على استخدام الـ TDataSet (التي تنحدر منها الـ TTable، الـ TQuery، والـ TStoredProc). ولكن، لقد أصبح ممكناً الآن أن تنشئ الـ TDataSet classes لا تحتاج إلى الـ BDE للتوصل إلى قواعد البيانات. يمكنك، مثلاً، أن تنشئ الـ TDataSet class الخاصة بك للوصول إلى Server قاعدة بيانات دون الخوض في الـ BDE - لا يجب حتى تركيب الـ BDE على نظام المستخدم النهائي.

يعتبر إنشاء الـ TDataSet class أمراً معقداً ويتطلب مهارات برمجة متطورة. إن class معنوية، بمعنى أنه لا يمكن تمثيلها كـ object. لاستخدام الـ TDataSet class الجديدة، يجب أن تستنبط class جديدة وقم بتوفير الـ code للـ methods المعنوية. كيف تفعل هذا؟ إن هذا يعتمد بدقة على Server قاعدة البيانات الخاص بك، وهو خارج نطاق هذا الكتاب.

إذا كنت تريد أن تنشئ الـ class الخاصة بك المشتقة بالتعامل مع الـ TBDE Dataset والـ TDB Dataset من الـ TDataSet، يمكنك أن تبدأ بدراسة كيف تقوم الـ Inprise Corporation بتطبيق الـ TBDE Dataset والـ TDB Dataset ومن الضروري أنك تحتاج أن تنشئ classes مشابهة تستدعي النظم الفرعية لبرنامج تشغيل قاعدة البيانات الخاص بك بدلاً من استخدام الـ BDE API. بعد إتمام هذه الخطوات، يمكنك إنشاء class شبيهة بالـ TTable class وتستخدمها في



تطبيقاتك. إن كل ال data-aware controls وال DataSource تعمل مع ال TDataSet class المشتقة الخاصة بك.

وهذا يشمل ال TClientDataSet، والتي يسهل تنفيذها إلى حد كبير عن ال TBDEDataSet:

### خاصية ال TField.IsBlob:

إن كل الحقول في جداول مجموعة البيانات يتم تمثيلها في ال Delphi ك objects من ال TField class وال classes ذات الأنواع الخاصة مثل ال TStringField وال TIntegerField تنحدر من ال TField class.

وال IsBlob، ترجع بـ True إذا كان ال TField object من نوع Blob (تذكر أن ال Blob هو Binary Large Object). فيتم تعريف ال function كما يلي:

```
class function IsBlob: Boolean; override;
```

في ال procedure يعرف ال TField parameter، يمكنك إستدعاء ال IsBlob لتحديد ما إذا كان الحقل Blob أم لا وهذا قد يسهل عملية عرض ال bitmap مقابل عرض النص وقيم أخرى، عى سبيل المثال، بعد إستخدام محرر ال Fields، يمكنك إختيار ال TField من ال Object Inspector وتنشئ event handler لـ مثل ال OnGetText. (لفتح محرر ال Fields، إضغط مرتين ال Table object، إضغط يميناً داخل نافذة محرر ال Fields، وإستخدم أوامر ال Add Fields وال New Field كما هو موضح في هذا الباب).

يستطيع ال event handler أن يتأكد ما إذا كان الحقل Blob ويتخذ العمل المناسب.

يمكنك برمجة ال event handler مع السطور التالية (هذا يفترض أنك قد كتبت ال DisplayAsBlob procedure لعرض ال TBlobField ك bitmap ربما ربما بمساعدة ال Image object في ال form:

```
procedure TMainForm.Table1YourFieldNameGetText(Sender: TField;
var Text: String; DisplayText: Boolean);
begin
```

```
if Sender.IsBlob then
begin
  DisplayAsBlob(Sender as TBlobField);
  Text := "";
else
  Text := Sender.AsString;
end;
```

### الفكر للمستخدم الخبير

● فكر في استخدام الـ DDB للتشارك في البيانات فيما بين تطبيقات قاعدة بيانات متعددة إن هذا يعتبر أكثر فعالية (وبالتأكيد هو مألوف أكثر للمستخدم) من كتابة بيانات للملفات القرص المؤقتة، والذي تعد عمل عادي. أنظر الباب السادس عشر لمعرفة تعليمات حول إقامة محادثات الـ DDE.

● كما ذكرنا عدة مرات في هذا الباب، قم دائماً بتسجيل Alias name باستخدام الـ Database Desktop، ولا تكتب في الـ code أى معلومات عن اسم المسار وبرنامج تشغيل قاعدة البيانات في تطبيقاتك. بالرغم من أن هذا يسهل فعله، إلا أن أسماء المسارات قد تعيقك عند تحريك ملفات قواعد البيانات إلى مواضع جديدة، أو محركات صلبة أخرى أو إلى الشبكات. إذا قمت باستخدام aliases للوصول إلى جداول قاعدة البيانات الخاصة بك، فإنك لن تفقد أثر مكانها، يجب أن يتم نقلها إلى محرك صلب أكبر وأحدث.

● ينشئ الـ Database Form Wizard الـ form الثانوية لتطبيق جديد. لإستخدام هذه الـ form كنافذة رئيسية للبرنامج، إضغط زر الـ Option التابع للـ Project Manager، وخصص الـ form تحت الـ Main form. يمكنك عندئذ إستخدام الـ Project Manage ليحذف الـ form الحالية للمشروع.

● إن برنامج تشغيل الـ Windows ODBC للـ Access قد تم تصميمه للإستخدام مع الـ Microsoft Office. لإنشاء تطبيقات قاعدة بيانات Access مع Delphi، يجب أن تحصل على الـ ODBC Desktop Driver التام.

إن برنامج تشغيل متوفر مع الـ Microsoft ODBC SDK وعلى الـ dBASE ، Btrieve ، Access ، Text ، Paradox ، Fox ، Excel .

• لتوزيع تطبيق قاعدة البيانات ، إنك تحتاج أن تزود ملف الـ .exe . التابع لبرنامجك (بالإضافة إلى أى ملفات مؤيدة للبيانات) والـ Inprise Corporation Database Engine . أنظر ملف الـ Redist.txt المتوفر مع Delphi لمعرفة معلومات حول الملفات اللازمة لتزويد المستخدمين النهائيين لك .

### المشروعات التى يمكنك تجربتها

١-١٧ : أكتب تطبيق لـ Table يحتوى على الأسم والعنوان . أو فكر فى أى تطبيق آخر بسيط إستخدم الـ Database Form Wizard لإنشاء محرر ومتصفح مناسبين .

٢-١٧ : قم بتوسيع قاعدة بيانات الأسم والعنوان الخاصة بك لتشمل رقم تعريف فريد لكل سجل . قم بمرمجة Tables إضافية وتطوير علاقات master-detail . على سبيل المثال ، بدلاً من تخزين البيانات الهامة مثل أعياد الميلاد والذكرى السنوية مباشرة فى سجلات الـ master ، قم بإنشاء Dates detail table مرتبط بأسماء الأصل . مع هذا الإعداد ، يمكن لأى اسم أن يكون له عدد غير محدود من التواريخ الهامة . وإنك توفر المساحة على القرص أيضاً ، بعدم حجز مساحة فى السجلات لا تريدها لتسجيل تواريخ .

٣-١٧ : إستخدم تطبيق الـ SQLPlay الخاص بهذا الباب لإكتشاف dataset أخرى فى قاعدة البيانات DBDEMOS المتوفرة مع Delphi . إختبر مع أوامر الـ SQL Select للبحث عن مسجلات ذات قيم حقول محددة .

٤-١٧ : قم بتجربة خيارات Database Form Wizard أخرى . على سبيل المثال ، يمكنك إستخدام الـ Export لإنشاء علاقات الـ master-detail ، ولإدخال Query objects لتنفيذ البحث .

## الملخص :

• تنقسم components قاعدة البيانات الخاصة بـ Delphi إلى مجموعتين :  
الـ Data Access التي توفر gateways و links بين التطبيق وقواعد البيانات ؛  
و Data Controls والتي تشمل data-aware controls والـ objects الأخرى  
لعرض وتحرير حقول البيانات ، وللملاحة في سجلات قاعدة البيانات .

• إن نسخ الـ Client-Server ، Professional ، Desktop ، Delphi  
تشمل الـ Borland Database Engine (BDE) ، الذي يوفر مجموعة كاملة من  
برمجة الـ API functions للعديد من نظم قواعد البيانات الشهيرة مثل الـ  
dBASE مثل الـ Microsoft Access و Btrieve . تشمل نسخة الـ  
Client-Server من Delphi كل مميزات نسخة الـ Desktop بالإضافة إلى أنها  
توفر الوصول إلى Server قاعدة البيانات مثل الـ Sybase ، Oracle ،  
Microsoft SQL Server ، Interbase ، Informix .

• إن الـ TDataSet الجديد لم يعد يعتمد على الـ BDE . يستطيع المبرمجين المتقدمين  
الآن أن يشتقوا class جديدة من الـ TDataSet لإستدعاء وظائف في Server قاعدة بيانات  
غريبة دون تركيب أو إستخدام الـ BDE . وهذا غير ضروري لأغلب التطبيقات .

• كبداية لإنشاء تطبيق قاعدة بيانات جديد ، إستخدم أمر الـ Database  
Form Wizard الموجود في قائمة الـ Database الخاصة بـ Delphi (لقد كان  
يلقب هذا الأمر بـ Database Form Expert وكان موجود في قائمة الـ Help) .  
ينشئ الـ Database Form Wizard الـ form التي تملك الـ Data Access و Data  
Controls والتي يمكنك تعديلها بعد توليد الـ form .

• الـ Table يعادل الـ dataset واحدة ، وهي تجمع لخلايا البيانات الذي يشكل  
صفاً وعموداً . إن الصفوف تعادل السجلات ؛ والأعمدة تساوي الحقول . تعتبر  
قاعدة البيانات مجموعة من table واحد أو أكثر . والـ Alias name هو اسم يخفي  
معلومات المحرك والشبكة والمسار واسم الملف للملفات وأدلة قاعدة البيانات الفعلية  
إستخدم دائماً والـ Alias name للإشارة إلى قواعد البيانات ؛ بهذه الطريقة تعمل  
تطبيقاتك بدون إعادة الـ compile إذا قمت بتحريك ملفات قاعدة البيانات إلى أدلة  
أخرى ، أو إذا قمت بتركيبها على شبكة بعد أن تختبر الـ code .

● تحتاج غالبية تطبيقات قواعد البيانات إلى Table و DataSource. يوفر ال Table بوابة إلى Table قاعدة البيانات (الذى يفضل الإشارة إليه بـ Alias name). يربط ال DataSource ال Table بواحد أو أكثر من ال Data Controls.

● إستخدم ال Database Desktop لإنشاء وتعديل Table قاعدة البيانات. يمكنك أيضاً رؤية وتحرير معلومات قاعدة البيانات بإستخدام هذا البرنامج التوضيحي، والذي هو بمثابة نهاية أمامية للـ BDE.

● توفر ال BDE مجموعة فرعية من أوامر ال SQL المعيارية (وهي Select، Insert، Update و Delete) ل formats قاعدة البيانات Paradox، dBASE، و Oracle format قواعد البيانات الأخرى.

يمكنك إستخدام اللغة الكاملة مع Servers آخرين إعتتماداً على تأييد ال SQL الخاصة بهم.

● ينفذ ال Query عبارات ال SQL. لأن components قاعدة بيانات Delphi توفر الكثير مما تستطيع ال SQL القيام به، فإنك غالباً تحتاج إلى إستخدام أمر ال SQL Select فقط لأداء البحث، مثلاً، ولتقصر ال dataset على السجلات التى تلائم المتغيرات المتنوعة.

● إن تطبيق قاعدة البيانات ال master-detail يربط إثنين من Tables قاعدة البيانات بإستخدام حقل مشترك مثل رقم العميل أو الرمز التجارى لسوق البورصة. إن تركيبة master-detail تتطلب زوجين من ال Table وال DataSource المتعلقة ببعضها.

● فكر فى وضع ال Table وال DataSource وال Data Access الأخرى الخاصة بك فى Data Module، ثم قم بتركيبها فى ال Object Repository الخاص بـ Delphi. تستطيع التطبيقات المتعددة عندئذ أن ترث أو تستخدم ال Data Module لتتمكن من الوصول إلى Tables قاعدة البيانات.

يحتوى الباب التالى على قصة برمجة قاعدة البيانات. سوف تتعرف على كيفية تطوير التقارير وال Charts بإستخدام معلومات قاعدة البيانات.

# الباب الثامن عشر

## تطوير الـ Charts والتقارير

محتويات هذا الباب:

- Components
- إنشاء الـ charts
- البداية مع الـ charts
- طباعة واصدار الـ charts
- Chart data sources
- إنشاء التقارير
- البداية مع التقارير
- طباعة عناوين العمود
- طباعة معلومات النظام
- تلخيص الاعمدة
- ترتيب بيانات التقرير
- طباعة التقارير في وقت التشغيل

يستطيع أى شخص ان يخزن بيانات فى الحاسوب ، ولكن استخلاص معلومات من قاعدة بيانات وتقديمها فى شكل قابل للاستخدام هو ما يتطلب مهارة حقيقية . ان رؤية البيانات هو ما يجعل المعلومات مفيدة وهامة ان واضعى برامج قاعدة البيانات ومديرو الانظمة قد امضوا غالباً كثير من الوقت فى تطوير التطبيقات لرؤية وطباعة الـ charts والتقارير عن أى نوع آخر من البرمجيات .

ولقد جعل Delphi هذه المهام سهلة كما ستعرف فى هذا الباب ، ان مكتبة ال TeeChart الخاصة بـ Delphi ، التى أنشأها ديفيد برنيدا ، توفر العديد العديد من انواع ال charts لعرض البيانات بصرياً . ومكتبة ال QuickReport 3 الخاصة بـ Delphi ، التى أنشأها آلان لوكيرت ، توفر مكتبة components متنوعة لتطوير كل انواع التقارير المطبوعة . يوفر Delphi 4 نسخ كاملة معيارية لكل مكتبة ، مع components مركبة على ال VCL palette ، جاهزة للاستخدام .

فى هذا الباب ، سوف اغطى تقنيات البرمجة المتوسطة والاساسية لمكتبات ال TeeChart وال QuickReport لإنشاء knock-your-socks-off printouts محترفة من معلومات مخزنة فى جداول قاعدة البيانات .

يأتى Delphi 4 مزوداً بطبوعات معيارية لمكتبات ال TeeChart وال QuickReport ، والتى يتم تغطيتها فى هذا الباب .

## Components

ان Delphi التى يتم تغطيتها فى هذا الباب هى :

● **Chart** : استخدم هذا ال components لإنشاء charts من بيانات مخزنة فى ملفات ، أو دخلت مباشرة فى برنامج . ال Data Controls : Palette .

● **DBChart** : استخدم هذا ال components لإنشاء charts من بيانات مخزنة فى Tables قاعدة البيانات . ال Data Controls : Palette .

● **DecisionGraph** : استخدم هذا ال components لإنشاء الرسوم البيانية من بيانات قاعدة البيانات . ال Decision Cube : Palette .

● **QRBand** : تتكون التقارير من bands ، كل واحدة منها تعتبر object من هذا ال components . ال QReport : Palette .

● **QRChart** : استخدم هذا ال components لإدخال charts فى تقارير قاعدة البيانات . ال QReport : Palette .

● **QRChildBand** : هذا ال components يتم استخدامه لإنشاء bands ممتدة مع components اخرى تتحرك أو تمتد هى الاخرى ، ولـ ال bands التى تمتد فوق صفحات متعددة . ال QReport : Palette .

## الباب الثامن عشر : تطوير الـ Charts والتقارير

● **QRDBImage** : استخدم هذا الـ components لإدخال Image في تقارير ، غالباً يكون واحداً مرتبطاً بحقل Blob في Table قاعدة البيانات . الـ QReport: Palette .

● **QRDBRichText** : استخدم هذا الـ components لإدخال Rich text object في التقرير . الـ QReport: Palette .

● **QRDBText** : استخدم هذا الـ components أغلب أنواع حقول Table قاعدة البيانات في هيئة نص . وهذا ينطبق ليس فقط على حقول النص ، ولكن أيضاً على الحقول الرقمية ، حقول البيانات ، حقول العملة ، والحقول الأخرى . الـ QReport: Palette .

● **QRExpr** : استخدم هذا الـ components لإنشاء تعبيرات مثلاً ، لحساب قيمة مدخل التقرير باستخدام حقول Table قاعدة البيانات أخرى . الـ QReport: Palette .

● **QRImage** : استخدم هذا الـ components لإدخال bitmap أو نوع آخر من صور مثل الايقونة في التقرير . يؤيد هذا الـ components كل أنواع الصور مثل الـ Image . الـ QReport: Palette .

● **QRLabel** : استخدم هذا الـ components لإدخال label ثابتة في تقرير الـ QReport: Palette .

● **QRMemo** : استخدم هذا الـ components لإدخال objects نص متعدد الاسطر في تقرير . الـ QReport: Palette .

● **QRPreview** : استخدم هذا الـ components لتوفر مراجعة على الشكل النهائي للتقرير ولكن بدلاً من استخدام هذا الـ components ، من السهل ان تستدعي الـ Preview method للـ QuickRep . الـ QReport: Palette .

● **QRRichText** : استخدم هذا الـ components لإدخال rich text object في تقرير . الـ QReport: Palette .



• **QRShape** : استخدم هذا components لإدخال objects جرافيك مثل المستطيل أو الدائرة في تقرير . ال QRReport: Palette .

• **QRSubDetail** : استخدم هذا components لربط dataset المتعددة في تقرير . إنك تفعل هذا لطباعة تقارير عن قواعد البيانات التي تستخدم تركيبة ال master-detail . ال QRReport: Palette .

• **QRSysData** : استخدم هذا components لإدخال system objects متنوعة مثل الوقت والتاريخ الحالي ، ارقام الصفحات ، والعناصر المتنوعة الاخرى في تقرير ، غالباً في header band . ال QRReport: Palette .

• **QuickRep** : استخدم هذا components لإنشاء تقرير جديد تقوم خاصية ال bitmap لل components بإدخال انواع متعددة من ال bitmas ، والتي تحمل حقول قاعدة البيانات والانواع الاخرى من ال objects التي سيتم طباعتها في التقرير التام . ال QRReport: Palette .

**ملحوظة:** ان كثير من ال QR (QuickReport) يتم إنشاءها من Data Control . اما ال components الاخرى يتم إنشاءها من controls غير controls قواعد البيانات . على سبيل المثال ، يمكن لل QRImage ان يدخل bitmap في تقرير ؛ ويفعل ال QRDBImage نفس الشيء ، ولكنه يتلقى بياناته من حقل Table قاعدة بيانات (في هذه الحالة ، يكون غالباً حقل Blob ، أو ال Large Object, field) .

Note

## إنشاء ال Charts مع ال TeeChart:

ان كل نوع من الانواع العديدة المختلفة من ال charts في ال TeeChart library يتم إنشاؤه من سلسلة من نقاط البيانات . وكل سلسلة هي نفسها object ينتمي لل chart . يوضح شكل (١٨-١) ال TeeChart Gallery dialog الذي تستطيع ان تختار منه نوع السلسلة التي تريد استخدامها في ال charts وكل سلسلة chart لها خيارات متعددة لتخصيص النتائج كما تشاءها . يمكنك إضافة labels ، تعديل اللون ، اختيار حدود متنوعة ، تحديد تأثيرات ال 3D ، والاختيار من بين خيارات اخرى عديدة .

**ملحوظة:** ان التوثيق الكامل للـ TeeChart library متوفر فى دليل تركيب Delphi . (لأننى استخدم نسخة سابقة على Delphi 4 ، فلا اعلم فى هذه اللحظة اين سيكون موضع ملف التوثيق ، ولكن فى Delphi 3 ، كان فى مسار دليل الـ TeeChart (Delphi 3). افتح ملف الـ TChartVx.doc (x تساوى رقم النسخة) باستخدام متصفح الـ Microsoft Word أو الـ Word و الـ online help الكاملة متاحة ايضاً- اختر أى TeeChart أو بنداً آخر فى نافذة Delphi form واضغط F1 لمزيد من المعلومات .

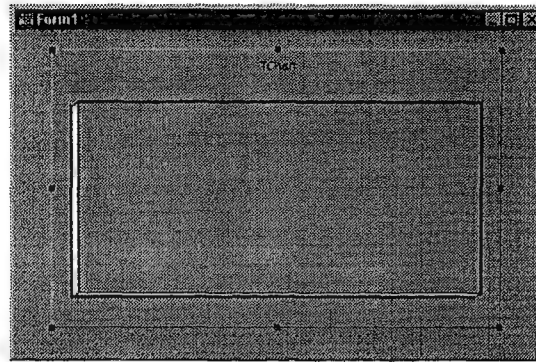


شكل (١٨-١) : لاختيار نوع الـ charts Series التى تريدها، اختر ايقونة من الـ TeeChart Gallery dialog . يمكن ان تعرض كل chart فى ثلاثية أو ثلاثة الابعاد

### البداية مع الـ charts:

ان اسهل الطرق للبداية فى استخدام الـ TeeChart library هو إنشاء charts بسيطة . إبدأ Delphi ثم إتبع هذه الخطوات (لا يجب عليك ان تحفظ المشروع):

- ١- إبدأ تطبيقاً جديداً .
- ٢- اضغط الـ Additional page tab على الـ VCL palette لـ Delphi .
- ٣- اضغط ايقونة الـ Chart ، ثم اضغط داخل نافذة الـ form لوضع Chart object جديد خالى . تبدو نافذة الـ form على شاشتك مثل شكل (١٨-٢) .



شكل (١٨-٢): استخدام الـ Chart لإنشاء Chart جديد خالى فى نافذة الـ form

**ملحوظة:** تنحدر الـ TChart class من الـ TPanel class التابعة لـ Delphi، ولذلك، يعتبر الـ Chart object هو حرفياً Panel تم زيادتها. هذا يعنى انك تستطيع ان تدخل objects اخرى مثل الـ Buttons والـ CheckBoxes فى Chart object. انظر الباب السابع لمزيد من المعلومات حول الـ Panel.

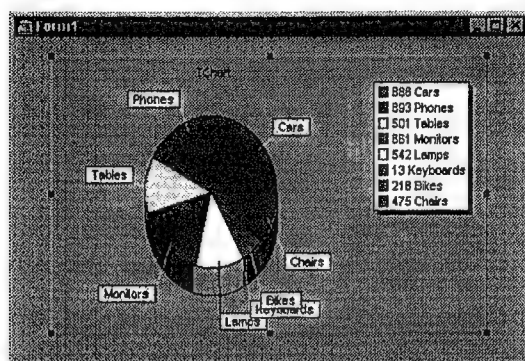


٤- اضغط ميمناً أثناء الاشارة بمؤشر الفأرة داخل الـ Chart. هذا يؤدي الى اظهار القائمة المحلية الخاصة بالـ Chart. اختر امر الـ Edit Chart... لفتح محرر الـ TeeChart. (وهناك طريقة اخرى لفتح المحرر وهى بالضغط مرتين على الـ Chart).  
٥- اضغط زر الـ Add للمحرر لإضافة مجموعة من سلسلة الـ chart. هذا يؤدي الى اظهار الـ TeeChart Gallery dialog الموضح فى شكل (١٨-١).  
اختر سلسلة الـ Pie من الـ Gallery، ثم اضغط OK واضغط Enter لإغلاق نافذة الـ Gallery dialog. قم بتحريك نافذة المحرر جانباً حتى يمكنك ان ترى الـ form والـ Chart. كما يوضح شكل (١٨-٣)، يعرض الـ TeeChart الـ Chart للنوع المختار مع نقاط بيانات عشوائية مضافة الى الـ Series.

**ملحوظة:** عندما تقوم بإنشاء chart جديد، تقدم مكتبة الـ TeeChart نقاط بيانات عشوائية حتى ترى نموذج من الناتج النهائى. وهذه البيانات لا يتم عرضها فى وقت التشغيل. عندما تقوم بتشغيل البرنامج، يستخدم الـ chart البيانات الخاصة بك، والتي يمكن ان تأتى من مصادر متنوعة. وهذه الخاصية



تساعدك على تصميم chart حسنة المظهر، وهى أيضاً طريقة رائعة لكى تتعرف على طريقك فى مكتبة الـ TeeChart.



شكل (١٨-٣): يوضح الـ TeeChart الـ Chart المختارة فى Delphi مع نقاط البيانات العشوائية

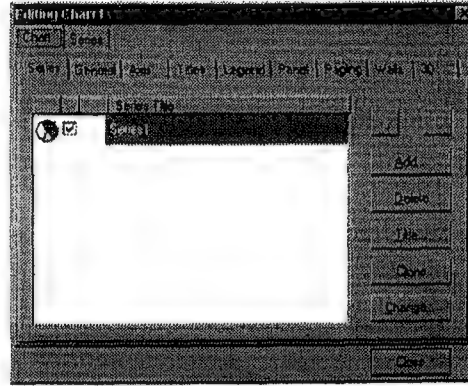
قم بتجربة انواع series اخرى فى الـ TeeChart Gallery [راجع شكل (١٨-١)]. اعد الخطوات السابقة بدءاً من الخطوة رقم ٤، واختر زر الـ Add لإضافة series objects إضافية. يمكن للـ chart ان يكون لها متعددة من نفس النوع- لمقارنة نقاط البيانات مثل اسعار سوق البورصة، مثلاً، مع خطوط متعددة فى grid x و y. ويستطيع الـ chart ايضاً تجميع series من انواع مختلفة- على سبيل المثال، لإظهار الـ bar and line charts فى نفس الـ grid، ربما للمقارنة بين المعلومات التاريخية والحديثة.

استخدم زر الـ Change لمحرر الـ TeeChart لتغيير نوع الـ chart المعروف لـ Series مختارة- على سبيل المثال، لتغيير bar chart لرسم نقطة. اضغط زر الـ Delete للمحرر لتحذف الـ Series المختارة (هذا يعرض dialog طلب تأكيد فى حالة ان تكون قد ضغطت الـ Delete بطريق الخطأ). قم بتجربة هذه الخيارات الآن حتى تعتاد عليها.

**ملحوظة:** كثير من خيارات محرر الـ TeeChart تختلف اعتماداً على نوع الـ series التى تختارها. للـ chart اذا كانت الـ objects الموجودة فى هذا الباب تختلف عما تراه على الشاشة فغالباً تكون قد اخترت نوع مختلف من Barchart series بدلاً من رسم الـ Line، مثلاً.



قم أيضاً باختيار بعض page tabs فى محرر الTeeChart [شكل (١٨-٤)]. هذا يفتح صفحات متنوعة فى نافذة المحرر لتخصيص الchart. كما ترى، باختيار بعض الجداول، توجد عشرات وعشرات من الخيارات لاختيار الالوان، انماط، الchart والسمات. جرب هذا: اضغط الGeneral tab وقم بإبطال الcheck box اختيار ال3D لتحول الchart الابعاد الى الchart ثنائى الابعاد. وكذلك جرب هذا: اضغط الLegend tab وقم بإبطال الVisible check box لإغلاق، والى تذكر قيم الpie chart [هذه هى النافذة الخلفية البيضاء الصغيرة الموضحة فى اعلى اليمين فى شكل (١٨-٣)].



شكل (١٨-٤): يوفر محرر الTeeChart خيارات عديدة لإنشاء وتعديل الcharts

لاحظ أن محرر الTeeChart يملك صفين من page tabs. وهذه ال tabs العليا تختار واحداً من المساحتين الكبيرتين فى المحرر. وهذه هى:

• **Chart:** اختر ال tabs العلوى الذى يحمل الlabeled Chart الخاص بالمحرر، ثم اختر الpage tabs على الصف الثانى لصنع تغييرات الى خصائص الchart. على سبيل المثال، اضغط الTitles tab وإدخل عناوين الchart. لاحظ أن تغييرات قد إنعكست فى الchart المعروض.

• **Series:** اضغط الSeries tab فى أعلى المحرر، ثم اختر الseries من قائمة اللائحة.

يمكنك عندئذ أن تختار ال tabs الصف الثانى لصنع التغييرات لخصائص الseries. اضغط الSeries الآن على الصف العلوى، اختر الSeries1، اضغط

## الباب الثامن عشر : تطوير الـ Charts والتقارير

الـ Marks tab على الصف الثانى ، واختر واحداً من الازار التالية تحت Style :  
اختر Value لإظهار قيم فقط البيانات فقط ، Percent لإظهار النسب المئوية ،  
Label & Percent لإظهار كلاً من labels النص والنسب المئوية لكل نقطة  
بيانات ، وما الى ذلك .

وهناك طريقة اخرى لاختيار خصائص الـ chart وهى باختيار الـ Chart  
(اغلق اولاً محرر TeeChart) ، ثم استخدم نافذة الـ Object Inspector  
لتخصيص خصائص الـ chart كما تفعل بـ Delphi الأخرى . ان أغلب الخصائص  
متاحة فى كلا من محررى TeeChart والـ Object Inspector . يمكنك استخدام  
أى method تريده ، ولكنك سوف تكتشف ان المحرر اسهل فى الاستخدام بسبب  
الطريقة التى ينظم بها الخصائص والخيارات فى صفحات . ان الـ Object  
Inspector يقدم اغلب البنود فى نافذة واحدة كبيرة ، وهو الأمر الذى يجعل من  
الصعب وضع العناصر التى تحتاجها .

**ملحوظة:** ان محرر الـ TeeChart له خيارات ومميزات عديدة بحيث  
يصعب تغطيتها فى هذا الباب وحده . ولكن بمتابعة الخطوات المقترحة  
الموضحة هنا يمكن تعلم كيفية استخدام خيارات اخرى بنفسك . وسوف  
اشرح ايضاً المزيد عن الخيارات الخاصة للـ charts فى الفصول التالية .

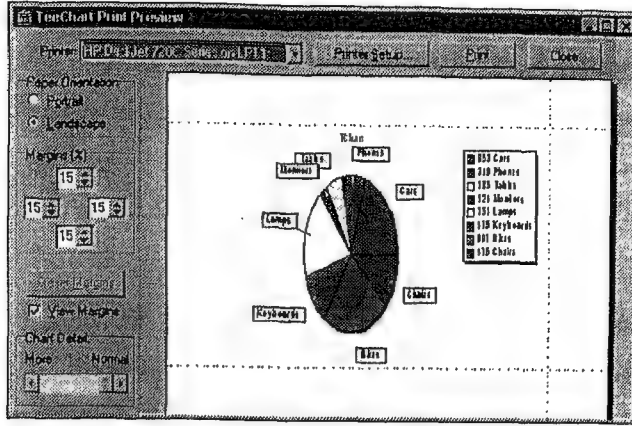
Note

### طباعة واصدار الـ charts:

تقدم الـ TeeChart مميزات طباعة ومراجعة الـ object الطباعة النهائى كاملة ،  
والتي يمكنك توفيرها للمستخدم النهائى فى وقت التشغيل . يمكنك استخدام نفس  
هذه المميزات فى Delphi لطباعة الـ charts . بالإضافة الى ذلك ، يمكنك حفظ أى  
charts فى ملفات جرافيك الدمج فى وثائق اخرى ، أو تنفيذ عبر الانترنت .

### طباعة الـ charts فى Delphi:

لطباعة الـ charts فى Delphi ، اضغط ميميناً داخل الـ Chart ، واختر امر الـ  
Print Preview... من القائمة المحلية . (اتبع الخطوات الموجودة فى الفصل السابق  
لإنشاء chart اذا كنت تريد واحدة) . يعرض الأمر نافذة الـ TeeChart Print  
Preview الموضحة فى شكل (١٨-٥) .



شكل (١٨-٥)، نافذة ال TeeChart Print Preview

توضح نافذة ال Print Preview الخريطة كما تظهر عندما يتم طباعتها. استخدم الخيارات المتنوعة لاختيار اتجاه ال Portrait أو ال Landscape، ولتعديل الهوامش، ولتعديل تفاصيل ال chart (التأثير بدقة يعتمد على نوع ال chart-امسك وارك زر ال Chart Detail scroll لترى كيف يغير الصورة).

يمكنك أيضاً استخدام الفأرة لضغط وسحب خطوط الهوامش المنقوطة وتحرك ال chart عبر الصفحة. جرب هذا الآن-اضغط الفأرة داخل نافذة مراجعة ال object النهائي واسحبها لتغير ال chart وحجمه.

عندما تنتهي من ضبط ال chart واختيار الخيارات (ايضاً اضغط زر ال Printer Setup... لاختيار خيارات الاخراج الخاصة بطابعتك)، اضغط زر ال Print للطباعة. يمكنك عندئذ ضغط زر ال Close لإغلاق نافذة ال Print Preview.

**تحذير:** عندما تغلق نافذة ال Print Preview، ترجع كل المواصفات الى البدائل الافتراضية حسب النظام. لا تغلق النافذة الا بعد الانتهاء من طباعة ال Chart.



### طباعة ال Chart وقت التشغيل:

يستطيع مستخدمو برنامجك ان يظهروا نافذة ال Print Preview، ويستخدموا كل الخيارات الموضحة فى الفصل السابق. لتجربة هذا، اصف ال

## الباب الثامن عشر : تطوير الـ Charts والتقارير

Button objectform ، واضغطه مرتين لإنشاء الـ OnClick event الخاص به .  
إنتقل الى نافذة محرر الـ Delphi code ، إبحث عن كلمة الـ implementation الأساسية . واضف عبارة uses كما يلي :

```
implementation
```

```
{ $R *.DFM }
```

```
uses
```

```
TeePrevi;
```

هذا يجعل (TeeChart Print Preview) متاحة للـ module . يمكنك الآن برمجة الـ Button event handler كما يلي :

```
procedure TForm.Button1Click(Sender: TObject);
```

```
begin
```

```
ChartPreview(Form1, Chart1);
```

```
end;
```

قم باستدعاء الـ ChartPreview procedure الخاص الـ TeePrevi unit وقم بتمرير الـ chart form (Form1 هنا) و (Chart1) على انها arguments هذا يعرض نافذة الـ Print Preview ، الموضحة في شكل (١٨-٥) . وتتولى الـ unit تماماً كل خيارات الطباعة-لا حاجة لكتابة أى closes اخرى . عندما يغلق المستخدم النافذة ، يستمر البرنامج بشكل طبيعي .

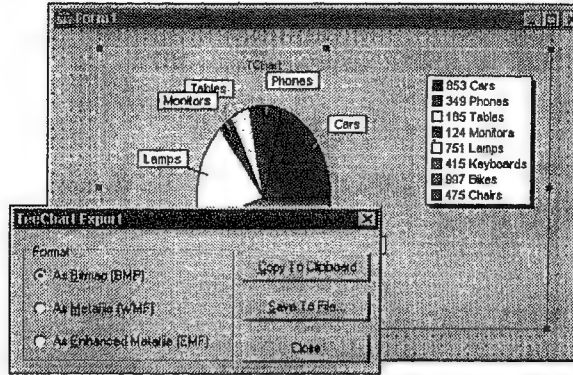
### تنفيذ الـ charts في Delphi

تنفيذ الـ charts في Delphi ، اضغط يميناً داخل الـ Chart objects لفتح القائمة TeeChart (إذا لزم الأمر ، قم بإنشاء chart) . اختار امر الـ Export Chart... الذى يعرض نافذة الـ dialog الموضحة في شكل (١٨-٦) .

ان الـ TeeChart Export dialog يقدم ثلاثة خيارات لنسخ chart الى الـ Windows clipboard ، أو لحفظ الـ chart فى ملف قرص . و الـ formats الثلاث هي :

● **As Bitmap (BMP)** : استخدم هذا الخيار لحفظ الـ chart فى ملف bitmap ، وحجم هذا الملف يعتمد على حجم الـ chart .





شكل (١٨-٦) استخدام الـ TeeChart Export dialog لتسخر صورة chart الى الـ Windows clipboard، أو لحفظها في ملف صورة

● **As Metafile (WMF)**: استخدم هذا الخيار لحفظ الـ chart في ملف meatfile. وهذا ليس صورة، ولكن مجموعة من الاوامر التي ترسم صورة تشبه بالضبط البرنامج الأصلي. والـ metafile تستخدم فقط مساحة قليلة على القرص، ويسهل قياسها بابعاد عرض جديدة. استخدم هذا الخيار اذا كانت صورتك تحتاج ان يتم رؤيتها من جانب مستخدمى الـ Windows 3.1.

● **As Enhanced Metafile (EMF)**: استخدم هذا الخيار لحفظ chart في ملف enhanced metafile تم زيادته. هذا يشبه ملف الـ metafile المعيارى، ولكن يمكن رؤيته فقط من قبل مستخدمى الـ Windows 95 و 98 و NT. وتعتبر ملفات الـ enhanced metafile التي تم زيادتها افضل من جوانب كثيرة، بما فى ذلك الدقة فى إعادة الانتاج والقياس، عن ملفات الـ metafile المعيارية. ان ملفات الـ enhanced metafile التي تم زيادتها تستخدم إحداثيات ذات ٣٢ بت؛ الـ metafile المعيارية تستخدم احداثيات ذات ١٦ بت. للحصول على افضل النتائج، استخدام ملفات الـ enhanced metafile الزائدة اينما كان ممكناً.

بعد اختيار واحد من خيارات التنسيق الثلاثة، اضغط زر الـ Copy to Clipboard لنسخ الصورة الى الـ Windows clipboard. يمكنك عندئذ ان تنتقل الى برنامج آخر مثل محرر النص وتلصق الصورة فى وثيقة (بافتراض ان البرنامج يستطيع التعامل مع الصور فى الـ form المختارة). اضغط زر الـ Save to File...

لإظهار الـ dialog الذى تستطيع استخدامه لإدخال اسم الملف المستهدف ، ولاختيار دليل الاخراج .

### تنفيذ الـ charts فى وقت التشغيل:

لتنفيذ الـ charts فى وقت التشغيل ، استدع واحداً من الـ TChart methods: الثلاثة التالية :

SaveToBitmapFile(const Filename: String);

SaveToMetafile(const Filename: String);

SaveToMetafileEnh(const Filename: String);

والى كل method ، قم بتمرير اسم الملف ، والذى قد يشمل معلومات عن اسم المسار والمحرك . يجب ان يتناسب امتداد اسم الملف مع نوع الملف - فهذا لا يتم إضافته بصورة تلقائية . على سبيل المثال ، لحفظ الـ chart للـ Windows bitmap ، استخدم عبارة مثل :

Chart1.SaveToBitmapFile('C:\Test.bmp');

**ملحوظة:** بعض النسخ من الـ online help للـ TeeChart تسمى الـ procedure السابق بـ SaveToBitmap وهو الصحيح هو .SaveToBitmapFile.

Note

استدع الـ functions الأخرى بنفس الطريقة . تقوم العبارتان التاليتان بحفظ الـ charts كملف metafile و كملف Windows enhanced metafile ثم زيادته . كما سبق ، إنها مسئوليتك ان تحدد امتداد اسم الملف الصحيح لنوع الملف الذى تنشئه :

Chart1.SaveToMetafile('C:\Test.wmf');

Chart1.SaveToMetafileEnh('C:\Test.emf');

### فهم مصادر بيانات الـ chart:

كما اوضحنا فى الفصول السابقة ، فمن السهل إنشاء chart مع مكتبة الـ TeeChart . ولكن اختيار افضل نوع لـ chart series معينة من نقاط البيانات هو عادة اصعب جزء فى مهمة عمل الـ chart . لإنشاء chart ناجحة فإنك تحتاج ان

تفكر جيداً فى مصدر بياناتك، والمعلومات التى يقدمها وهدفك من توفير تمثيل بصرى لتلك المعلومة.

لكى نساعدك فى البدء فى تطوير ال charts حقيقية باستخدام ثلاثة انواع اساسية من مصادر البيانات:

● **Program data**: استخدام مصدر البيانات هذا لإنشاء charts باستخدام بيانات تكتبها مباشرة فى ال source code للبرنامج. ويمكن تثبيت هذه البيانات لعرض chart لمعلومات تاريخية مثلاً- أو يمكن حسابها باستخدام functions البرنامج.

● **File data**: استخدام مصدر البيانات هذا لإنشاء chart بيانات مخزنة فى ملف قرص. ويرجع نوع الملف إليك. فقد يكون ملف لبيانات تم إنزالها من الانترنت، أو يمكن ان يكون ملف نص ذو معلومات تدخلها انت أو تقصصها وتلصقها من وثيقة اخرى.

● **Database data**: استخدام مصدر البيانات هذا لإنشاء charts باستخدام بيانات تم تحميلها من Table قاعدة بيانات. وكحد ادنى، إنك تحتاج ايضاً لاستخدام Table components و DataSource لتوصيلها الى قاعدة بيانات. (انظر الباب السابع عشر لمزيد من المعلومات عن components قاعدة بيانات (Delphi).

يوضح الفصل التالى كيفية برمجة ال charts باستخدام بيانات من كل من هذه المصادر الثلاثة. والبرامج فى كل فصل ايضاً توضح تقنيات charts إضافية.

### charts من بيانات برنامج:

ان إحدى الطرق لتوفير بيانات ل chart هي كتابتها مباشرة فى ال source code للبرنامج. أو، يمكنك استخدام هذا ال method لعرض معلومات تم توفيرها بواسطة procedures متنوعة مثل تشغيل functions النظام. لهذه الانواع من البيانات، استخدم ال Chart على ال Additional palette.

قم باسقاط Chart على form، اضغطه مرتين واختر Edit Chart... لفتح محرر ال TeeChart. اضغط زر ال Add لإنشاء واحداً أو أكثر من ال Series

## الباب الثامن عشر : تطوير الـ Charts والتقارير

ولاختيار نوع الـ series objects الذى يتم عرضه. يمكنك ايضاً اختيار خيارات متنوعة للـ charts فى هذا الوقت.

بعد الانتهاء من تصميم الـ charts، اضف عبارات برنامج لتزودها ببيانات حقيقية. لإدخال البيانات، استدع الـ Add methods، أو AddY، أو AddX، أو AddXY للـ series objects الخاصة بك لإضافة نقاط بيانات. على سبيل المثال، تضيف العبارات التالية AValue (بافتراض انها متغير مزدوج) للـ Series1، وتحدد label نقطة البيانات هذه بالـ 'Value' string، وتحدد ان الـ Chart يجب ان يختار لوناً ملائماً:

```
Series1.Add(AValue, 'Value', clTeeColor);
```

وبدلاً من الـ clTeeColor، يمكنك استخدام أى قيمة TColor مثل الـ clBlue أو الـ clRed، أو يمكنك تحديد اللون بالقيم العشرية. (لمزيد من المعلومات راجع استخدام الـ TColor فى online help الخاصة بـ Delphi، وكذلك انظر الباب الثالث عشر، "تطوير تطبيقات الجرافيك").

**على القرص المدمج:** ان التطبيق MemInfo على القرص المدمج فى دليل Source\MemInfo يوضح كيفية إضافة بيانات برنامج الى chart. يوضح شكل (١٨-٧) البرنامج، الذى يعرض الـ bar chart



لذاكرة البرنامج تستخدم كما هى واردة من قبل الـ System.GetHeapStatus function الخاصة بـ Delphi. لتشغيل البرنامج، قم بتحميل ملف مشروع الـ MemInfo.dpr فى Delphi واضغط F9. عندما تظهر نافذة البرنامج، اضغط زر الـ ShowInfo لعرض قيم استخدام ذاكرة متنوعة. يمكنك إضافة form هذا البرنامج الى مشروعاتك لإزالة مشكلات الذاكرة، أو لمجرد مراجعة استخدام الذاكرة. توضح القائمة (١٨-١) الـ source code للبرنامج.

القائمة (١٨-١): MemInfo\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

---

Windows, Messages, SysUtils, Classes, Graphics,  
Controls,  
Forms, Dialogs, StdCtrls, TeEngine, Series, ExtCtrls,  
TeeProcs, Chart, Buttons;

```

type
  TMainForm = class(TForm)
    Chart1: TChart;
    Button1: TButton;
    Series1: TBarSeries;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

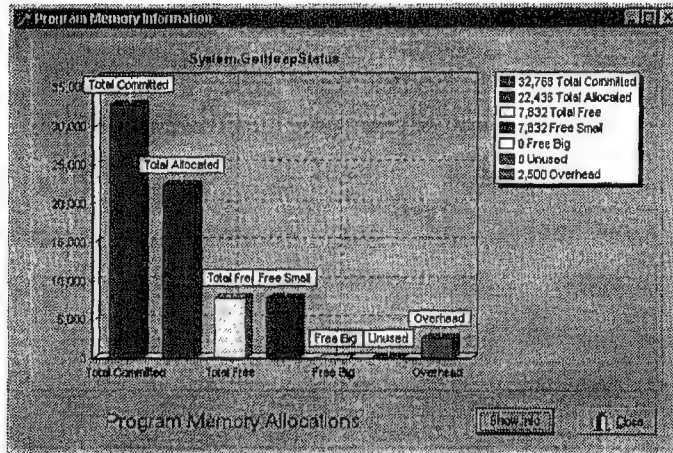
implementation

{$R *.DFM}

var
  HeapStatus: THeapStatus;

procedure TMainForm.Button1Click(Sender: TObject);
begin
  HeapStatus := System.GetHeapStatus;
  with Series1, HeapStatus do
    begin
    // Add(TotalAddrSpace, 'Total address space',
    clTeeColor);
  
```

```
// Add(TotalUncommitted, 'Total Uncommitted',
  clTeeColor);
  Add(TotalCommitted, 'Total Committed', clTeeColor);
  Add(TotalAllocated, 'Total Allocated', clTeeColor);
  Add(TotalFree, 'Total Free', clTeeColor);
  Add(FreeSmall, 'Free Small', clTeeColor);
  Add(FreeBig, 'Free Big', clTeeColor);
  Add(Unused, 'Unused', clTeeColor);
  Add(Overhead, 'Overhead', clTeeColor);
end;
end;
end.
```



شكل (١٨-٧): تطبيق MemInfo يوضح استخدام البرنامج للذاكرة وكذلك يعرض كيفية إضافة بيانات برنامج إلى الـ Chart

للحصول على قيم استخدام الذاكرة، يستدعى برنامج الـ MemInfo والـ function GetHeapStatus في الـ System unit الخاصة بـ Delphi. تقوم الـ function بإرجاع سجل من نوع الـ THeapStatus، وهو معرف كمتغير في قطاع الـ implementation للـ MemInfo. تملأ العبارة التالية سجل الـ HeapStatus بقيم مثل الـ TotalCommitted والـ TotalAllocated، والتي تمثل استخدام ذاكرة البرنامج. (راجع الـ online help الخاصة بـ Delphi لمعرفة المزيد من المعلومات من كل متغير في سجل الـ THeapStatus).

```
HeapStatus := System.GetHeapStatus;
```

لإضافة القيمة التى تم إدخالها الى ال Series1 الخاص بال chart ، يقوم ال  
OnClick الخاص بزر ال Show Info باستدعاء ال Add method . على سبيل  
المثال ، هذه العبارة تضيف قيمة ال TotalFree الى الخريطة :

```
Add(TotalFree, 'Total Free', clTeeColor);
```

هذا يؤدى الى إنشاء bar جديد فى القيمة الموجودة فى ال HeapStatus ال  
TotalFree يحدد labels لل bar من خلال string . وقيمة اللون clTeeColor  
تخبر ال color ان يختار اللون بصورة تلقائية لنقطة البيانات هذه . ويتم إضافة نقاط  
البيانات الأخرى بنفس الطريقة .

**ملحوظة:** لا يعرض ال MemInfo لإثنين من قيم ال  
GetHeapStatus ، وهى TotalAddrSpace و TotalUncommitted . فهذه تعتبر قيم نظام عامة ، والتى لا ترتبط  
مباشرة بقيم ذاكرة البرنامج الأخرى الموجودة فى الخريطة . وهذا مثال جيد على متى  
تكون المعلومات الكثيرة جداً أكثر ضرراً ، لذا فقد حذفت العبارتين الواقعتين  
بالقرب من بداية ال Button .

ان ال Add method المستخدم فى ال MemInfo يعتبر مناسباً لل bar  
charts البسيطة . وهو أيضاً ملائم لإنشاء ال pie charts لان هذه توضح البيانات  
كنسب مئوية من الكل - لمقارنة الكثافات السكانية للمدن ، مثلاً ، أو لإظهار عدد  
المنتجات فى مخزن مصنع . والانواع الأخرى من ال charts مثل راسمات الخطوط  
تتلقى المعلومات المرسومة على grid احداثيات ال x وال y . لهذه الانواع من ال  
charts احداثيات قد يكون لديك فى بعض الاحيان قيم x أو y ، أو قد يكون لديك  
كليهما . لإضافة نقاط البيانات لهذه الانواع من ال series ، استدع ال AddXY  
procedures أو AddY أو AddX . تضيف العبارة التالية YValue (قيمة  
مزدوجة من المفترض ان تكون معرفة من قبل البرنامج) ، وتعطى محور ال Y بطاقة  
تحمل 'Y-Label' string ، وتلون العرض الخاص بهذه البيانات باللون الاحمر :

```
Series1.AddY(YValue, 'Y-Label', clRed);
```

## الباب الثامن عشر : تطوير الـ Charts والتقارير

اجداثية الـ x لنقطة البيانات هذه يتم حسابها بصورة تلقائية باستخدام القيم المختارة مع page خيارات الـ ChartAxis الخاصة بمحرر الـ TeeChart . بنفس الطريقة ، اذا كان لديك قيمة x ، استدع الـ AddX لإنشاء نقطة بيانات جديدة للـ series :

```
Series1.AddX(XValue, 'X-Label', clWhite);
```

اخيراً ، اذا كان لديك إثنين من القيم للـ chart كمواضع x و y فى الـ grid ، استدع الـ AddXY باستخدام عبارة مثل هذه :

```
Series1.AddXY(XValue, YValue, 'X-Label', clBlue);
```

### Charts من بيانات ملف:

ان مصدر البيانات الذى يعد أكثر إنتشاراً للـ charts يأتى من معلومات مخزونة فى ملفات والمشكلة الرئيسية فى قراءة هذه البيانات هى معرفة format الملف . فقد يكون ملف نص تم انزاله من على شبكة الانترنت ، أو قد يحتوى على قيم مخزنة فى صورة binary اذا عرفت أو استطعت تحديد format الملف ، فيمكنك كتابة procedure لتحميل البيانات فى الذاكرة ثم استدعاء الـ AddX ، أو الـ AddY ، أو الـ AddXY methods لإدخال نقاط بيانات فى chart series.

### على القرص المدمج: وكعرض واقعى للمشكلات المتضمنة فى قراءة

ملفات بيانات ، فإن التطبيق MayTemp على القرص المدمج فى دليل Source\MayTemp يعرض رسماً بيانياً للبيانات الرقمية التى



قمت بانزالها من على موقع شبكة الانترنت <http://www-mfl.nhc.noaa.gov> . هذا يتوصل الى مكتب الـ National Weather Service (أو الخدمة الوطنية للمناخ) فى ميامى بولاية فلوريدا ، التى توفر تنبؤات عن حالة الطقس ، charts ومعلومات اخرى . فى هذه الحالة ، كنت مهتماً . بيانات المناخ فى منطقة Key West بولاية فلوريدا لشهر مايو ، وبخاصة متوسط درجات الحرارة العظمى والصغرى . وهذه البيانات موجودة فى ملف [climate/Key-West\\_May\\_Climat.html](http://climate/Key-West_May_Climat.html) والموضح فى الـ form الاساسية



في شكل (١٨-٨) كما هو معروض على نظامي باستخدام ال Internet Explorer. كما ستدرك ، ان ال Internet محملة ببيانات متمعة مثل هذه .

Key West May Client - Microsoft Internet Explorer (Working Offline)

Address: http://www.ndbc.noaa.gov/climate/Key-West\_May\_Climat.html

## KEY WEST, FLORIDA

MAY Daily Normal and Record Temperatures  
Monthly Normal and Record Precipitation  
January 1871-1996

Date	Normal Low @	Year	Record Low	Year	Coldest Maximum	Year	Year
1	83	74	90	1873	64	1992	74
2	84	74	89	1953	63	1977	76
3	84	75	91	1873	66	1992	76
4	84	75	90	1956	66	1925	74
5	84	75	91	1873	64	1992	72
6	84	75	90	1946	67	1992	75

شكل (١٨-٨): بيانات المناخ الاصلية من ال National Weather Service لمنطقة Key West بولاية Florida لشهر مايو (وبطاقة January (يناير) هي خطأ بالتأكيد) من ١٨٧١ الى ١٩٩٦

**ملحوظة:** ان القراء ذوى الابصار الحادة قد يلحظون ان البيانات الاصلية تحمل البطاقة "MAY" ولكن لها عنوان جانبي - "January 1871 - 1996". من الواضح ان هذا خطأ (فهذه بيانات May، وليس January)، وهذا ما ارسلت به لل NWS. وهذا ايضا يوضح ان، عند إنزال بيانات من على الانترنت، من الافضل ان تختبر صلاحية المعلومات التى يتم تلقيها.

**على القرص المدمج:** ان الحصول على البيانات الخام كان هو الجزء السهل - ان نقل هذه البيانات من على الانترنت الى ملف ثم الى برنامج Delphi للعرض كرسوم بياني بصرى يتطلب مزيداً من العمل. لقد اخترت اولاً البيانات الخام فى نافذة ال Internet Explorer، ضغطت Ctrl+C

Note



## الباب الثامن عشر : تطوير الـ Charts والتقارير

لنسخ النص الى لوحة المصقات، ثم قمت بفتح الـ Windows Notepad utility.  
ان ضغط Ctrl+V يلصق المعلومات الى الـ Notepad. بعد حذف النص الزائد  
قمت بحفظ البيانات في ملف نص يحمل اسم KeyWestMayClimate.txt.  
وهذا الملف، والموضح هنا في القائمة (١٨-٢)، موجود على القرص المدمج في  
دليل SourceData. (وكذلك موجود في هذا الدليل كمرجع ملف الـ  
Key-West\_May\_Climat.html، والذي يمكنك تحميله في متصفح الانترنت  
الخاص بك اذا اردت التدريب على ارسال البيانات بنفسك).

### القائمة (١٨-٢): Data\KeyWestMayClimate.txt

1	83	74	90	1873	64	1992	74	1988	80	1964
2	84	74	89	1953	63	1977	76	1925	79	1964
3	84	75	91	1873	66	1992	76	1897	80	1994
4	84	75	90	1956	66	1925	74	1989	80	1984
5	84	75	91	1873	64	1992	72	1889	80	1991
6	84	75	90	1946	67	1992	75	1911	81	1984
7	84	75	89	1956	66	1954	76	1903	80	1991
8	84	75	91	1873	65	1988	74	1898	80	1991
9	84	75	92	1873	64	1992	76	1928	80	1978
10	85	75	90	1873	66	1960	75	1891	80	1995
11	85	76	90	1873	64	1944	77	1891	80	1995
12	85	76	90	1926	68	1944	77	1900	80	1996
13	85	76	90	1995	66	1888	76	1900	81	1995
14	85	76	90	1878	65	1917	74	1900	81	1995
15	85	76	90	1991	68	1932	76	1917	81	1994
16	85	76	90	1991	66	1951	76	1977	82	1994
17	85	76	92	1878	66	1951	78	1880	81	1989
18	85	76	92	1881	67	1951	77	1904	81	1995
19	85	77	92	1881	68	1917	77	1904	82	1995
20	86	77	91	1886	68	1911	76	1875	81	1995
21	86	77	91	1935	68	1901	74	1875	80	1885
22	86	77	92	1935	68	1940	76	1921	82	1985
23	86	77	92	1873	69	1940	77	1892	80	1995

24 86 77 92 1873 68 1900 78 1883 82 1985  
 25 86 77 91 1989 70 1913 78 1982 82 1995  
 26 86 77 91 1989 68 1992 81 1879 81 1995  
 27 86 77 90 1989 70 1894 79 1880 81 1995  
 28 86 77 92 1953 68 1916 77 1901 81 1924  
 29 86 77 91 1952 68 1926 78 1918 81 1995  
 30 86 78 92 1949 67 1932 76 1960 82 1915  
 31 87 78 93 1881 70 1934 79 1960 82 1985

**على القرص المدمج:** ان البيانات الموضحة هنا عبارة عن text form تعتبر نموذجية . فى هذه الحالة ، تخبرنا البيانات الاصلية عن صفه القيم ، لذا لا يصعب قراءتها فى متغيرات برنامج . ولان هذا يعتبر ملف نص ، فإن ال Read procedure الخاص بال Object Pascal يعد افضل طريقة لتحميل البيانات وعرضها بصرياً . يوضح شكل (١٨-٩) ال chart التام ، والمعروضة من خلال تطبيق ال MayTemp على القرص المدمج فى دليل Source\MayTemp . قم بتحميل ملف مشروع ال MayTemp.dpr و Delphi واضغط F9 لل compile والتشغيل . توضح القائمة (١٨-٣) ال source code للبرنامج .



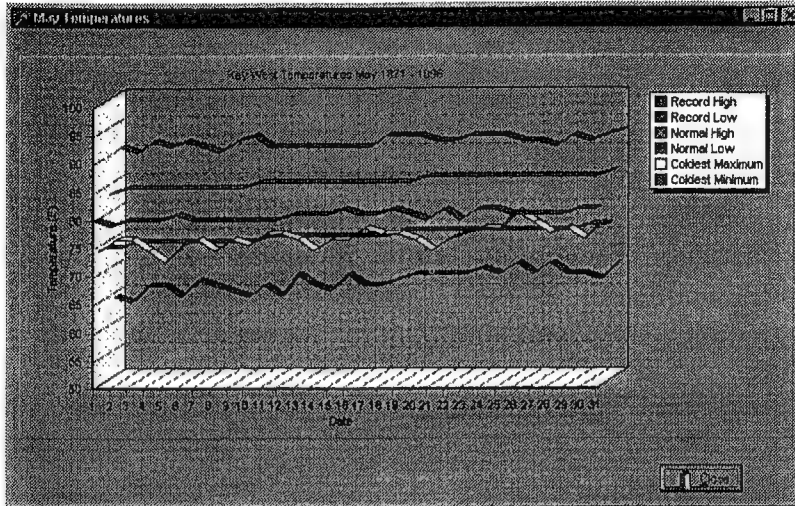
**ملحوظة:** يقرأ برنامج ال MayTemp ملف البيانات KeyWestMayClimate.txt الموجود فى دليل ال Source\Data على القرص المدمج . اذا فشل البرنامج فى التشغيل بشكل سليم ، فتأكد من ان الدليل الحالى هو Source\MayTemp ، أو قم بتعديل اسم المسار فى ثابت ال FileName الخاص بالبرنامج .



يستخدم ال MayTemp تقنيات Object Pascal معيارية لفتح ملف نص وقراءة معلوماته . لفعل هذا فى وقت التشغيل ، يستخدم البرنامج ال OnActivate للform . هذا يضمن ان تكون البيانات قد تم تحميلها وإضافتها بطريقة سليمة الى chart series objects ، عندما يبدأ البرنامج ، ولكن قبل ان تصبح النافذة مرئية . ان TMainForm.FormActivate ، يعرف متغيرات صحيحة لكل قيمة بيانات-متغير واحد لكل بند على كل صف [انظر شكل (١٨-٢)] . لقراءة بيانات الملف فى هذه المتغيرات ، يعرف ال procedure أيضاً متغير ملف :

var

F: TextFile;



شكل (١٨-٩): الـ chart التام، توضح بيانات درجات الحرارة من المعلومات الخام الموجودة بالقائمة (١٨-٢)

القائمة (١٨-٢): MayTemp\Main.pas

unit Main;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics,  
Controls,  
Forms, Dialogs, TeEngine, Series, ExtCtrls, TeeProcs,  
Chart,  
StdCtrls, Buttons;

type

TMainForm = class(TForm)  
Chart1: TChart;  
Series1: TLineSeries;  
Series2: TLineSeries;

```

Series4: TLineSeries;
Series5: TLineSeries;
Series6: TLineSeries;
Series3: TLineSeries;
BitBtn1: TBitBtn;
procedure FormActivate(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;

var
MainForm: TMainForm;

implementation

{$R *.DFM}

{ Change the following path name if your data file is
in another location. The path name is relative to the
current directory, which is assumed to be at the
same level as \Data\. }

const
FileName = '..\Data\KeyWestMayClimate.txt';

{ The following procedure is called when the form is first
activated. At that time, the program opens the data file,
reads its values, and adds them to the chart's series
objects. }

procedure TMainForm.FormActivate(Sender: TObject);
var
F: TextFile;           // File variable
Date: Integer;         // First column in file data

```

```

NormalHigh : Integer;      // Next column
NormalLow : Integer;       // and so on ...
RecordHigh : Integer;
RecordHighYear : Integer;
RecordLow : Integer;
RecordLowYear : Integer;
ColdestMaximum : Integer;
ColdestMaximumYear : Integer;
WarmestMaximum : Integer;  // ... down to the
WarmestMaximumYear : Integer; // Last column in
file data
begin
AssignFile(F, FileName); // Initialize file variable
Reset(F);                // Open the file
while not Eof(F) do      // Loop until the end of the file
begin
Read(F,                  // Read one row of data
Date,                   // into the individual variables.
NormalHigh,
NormalLow,
RecordHigh,
(continued)
Listing 18-3 (continued)
RecordHighYear,
RecordLow,
RecordLowYear,
ColdestMaximum,
ColdestMaximumYear,
WarmestMaximum,
WarmestMaximumYear); // End of Read
statement

```

{ One row of file data has been loaded at this point. The following statements add the data points to each of the line chart's six series objects. The empty string

arguments can be used to label data points. These strings aren't used here because the X axis of this sample chart already shows day values (1, 2, ..., 31). }

```
Series1.AddXY(Date, NormalHigh, ", clTeeColor);
Series2.AddXY(Date, NormalLow, ", clTeeColor);
Series3.AddXY(Date, RecordHigh, ", clTeeColor);
Series4.AddXY(Date, RecordLow, ", clTeeColor);
Series5.AddXY(Date, ColdestMaximum, ",
clTeeColor);
Series6.AddXY(Date, WarmestMaximum, ",
clTeeColor);
```

```
end;
end;
```

```
end.
```

هناك عبارتان تبدآن هذا المتغير باستخدام الـ FileName string الثابت ، ويفتحان الملف للإعداد لقراءة بياناته :

```
AssignFile(F, FileName);
Reset(F);
```

بعد هذه الخطوات ، تستمر عبارة while فى التنفيذ حتى يتم تحميل كل صفوف البيانات فى متغيرات البرنامج . وهذه الجزئية ، على سبيل المثال ، توضح كيف يقوم البرنامج بتحميل القيمة الاولى فى كل صف ، ممثلاً البيانات (١ ، ٢ ، ...٣١) :

```
while not Eof(F) do
begin
  Read(F,
    Date,
    ...
end;
```

## الباب الثامن عشر : تطوير الـ Charts والتقارير

يتم تحميل المتغيرات الاخرى بطريقة مشابهة ، مع عبارة Read واحد ، وذلك بذكرها منفصلة بواسطة الفاصلات .

**فكرة:** استخدم الـ Read procedure الخاص بالـ Pascal لقراءة قيم فردية على نفس خط النص الاصلى فى متغيرات منفصلة . استخدم الـ Readln لقراءة سطر بأكمله كـ string .

بعد تحميل كل صف بيانات ، يقوم البرنامج بادخال أغلب القيم فى chart series . فى هذه الحالة ، يكون لكل خط معروض فى الرسم البيانى النهائى ، [انظر شكل (١٨-٩)] . يتم اضافة كل نقطة بيانات باستدعاء الـ TeeChart AddXY method على سبيل المثال ، العبارتان التاليتان تدخلان قيم NormalHigh و NormalLow فى series objects الخاصة بهما على التوالى :

```
Series1.AddXY(Date, NormalHigh, " , clTeeColor);  
Series2.AddXY(Date, NormalLow, " , clTeeColor);
```

ان الأولى Date argument ، وهى نفسها لكلاً من نقطتى البيانات - يمثل محور الـ x فى هذه الحالة يوم فى الشهر الـ argument التالية هى القيمة التى يجب استخدامها . لنقطة البيانات هذه . ان الـ null string لا حاجة له لان chart يعرض بالفعل قيم درجات حرارة على طول محور الـ y . و argument الاخيرة تخبر الـ chart ان يستخدم اللون الذى تم برمجته فيما سبق لنقطة البيانات هذه .

**فكرة:** ان البيانات العشوائية الخاصة بالـ TeeChart للـ chart الموجود فى تطبيق الـ MayTemp لا تنتج الـ chart فى وقت التصميم . وهذا يحدث لان القيم العشوائية تقع خارج نطاق إحداثيات الـ x والـ y المحددة والتى اريد استخدامها للـ chart . لإغلاق توليد البيانات العشوائية ، قمت بفتح محرر الـ TeeChart ، وضغطت الـ Series page tab ، واخترت كل series object من قائمة اللائحة . لكل series ، قمت باختيار الـ Data Source page tab ثم الـ No Data من قائمة اللائحة لخيارات مصدر البيانات . قد تريد ان تفحص هذه المنطقة من محرر الـ TeeChart فى حالة ما اذا احتجت ان تغلق توليد البيانات العشوائية لـ charts .



## Charts من بيانات قاعدة البيانات:

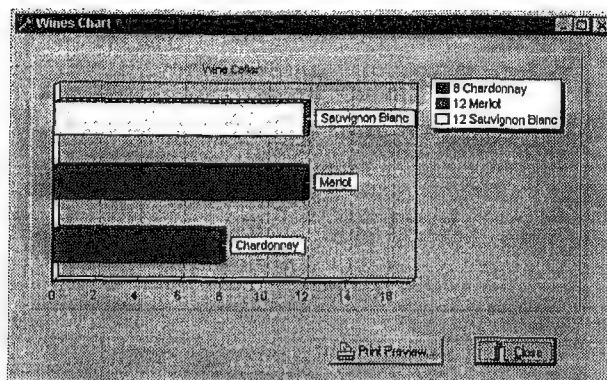
والطريقة الثالثة والاخيرة للحصول على بيانات chart هي تحميل قيم حقل من جداول قاعدة بيانات. وهذه التقنية تستخدم components مثل ال Table وال DataSource، الموضحة فى الباب السابع عشر، لفتح Table قاعدة بيانات. ولكن بدلاً من ال Chart component المستخدم حتى الآن، فلتحميل بيانات من جدول قاعدة بيانات، استخدم ال DBChart component الموجود على ال Data Controls palette الخاصة بـ Delphi. يعمل ال DBChart تماماً مثل ال Chart، باستثناء ان بياناته تأتى من جدول قاعدة بيانات. وكل السمات والخيارات الأخرى هي نفسها المذكورة فى الفصول السابقة.

**على القرص المدمج:** على القرص المدمج، يوضح تطبيق ال WinesChart كيفية إنشاء chart من معلومات قاعدة بيانات. يوجد هذا البرنامج فى دليل ال Source\WinesChart. يوضح شكل (١٨-١٠) عرض البرنامج باستخدام قاعدة بيانات ال Wines فى دليل ال Source\Data على القرص المدمج. توضح القائمة (١٨-٤) ال source code للبرنامج.



**ملحوظة:** اذا لم يعمل ال WinesChart على نظامك، فإن السبب غالباً يكون نقص ال WINES Alias name. لإنشاء هذا ال Alias name، اختر Tools\Database Desktop، ثم فى برنامج الحاسب المكتسب، اختر Tools\Alias Manager.... اضغط زر ال New، وإدخل WINES على انه ال Alias name. حدد الدليل المرتبط بدليل Source\Data\Wines من على القرص المدمج. اضغط Keep New ثم اختر OK. اجب بـ Yes، عندما تسأل عما اذا كنت تريد حفظ ال Alias name عامة. يجب ان يكون قادراً الآن على تشغيل ال WinesChart بنجاح. انظر الباب السابع عشر لمزيد من المعلومات حول تحديد واستخدام ال Alias name لقاعدة البيانات.





شكل (١٨-١٠): عرض وقت التشغيل لبرنامج الـ WinesChart يوضح كيفية قراءة معلومات من جدول قاعدة بيانات وعرضها كـ chart

القائمة (١٨-٤): WinesChart\Main.pas

unit Main;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics,  
Controls, Forms, Dialogs,  
StdCtrls, Buttons, TeEngine, Series, ExtCtrls, TeeProcs,  
Chart, DBChart,  
Db, DBTables;

type

TMainForm = class(TForm)

Table1: TTable;

DataSource1: TDataSource;

DBChart1: TDBChart;

Series1: THorizBarSeries;

BitBtn1: TBitBtn;

BitBtn2: TBitBtn;

procedure BitBtn2Click(Sender: TObject);

private

```

{ Private declarations }
public
{ Public declarations }
end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

uses
  TeePrevi;

procedure TMainForm.BitBtn2Click(Sender: TObject);
begin
  ChartPreview(MainForm, DBChart1);
end;

end.

```

لإنشاء تطبيق الـ WinesChart إتبع الخطوات التالية :

١- قم بإضافة Table و DataSource على form ، مع DBChart object . كذلك إدخال إثنين من الأزرار ، تحمل ...labeled Print Preview في Close كما في شكل (١٨-١١) ، الذي يوضح نافذة الـ form المنتهية في Delphi .

٢- حدد خاصية الـ DatabaseName للـ Table1 بـ WINES حدد الـ TableName بـ Wines.db . ثم ، غير الـ Active لتصبح True .

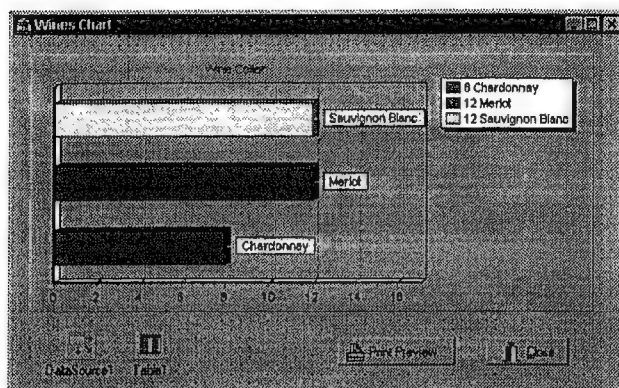
٣- حدد خاصية الـ DataSet للـ DataSource1 بـ Table1 .

٤- اختر DBChart1 ، واضغط الفأرة يميناً لإظهار القائمة المحلية للـ

TeeChart . اختر Edit Chart... لعرض محرر الـ TeeChart .

## الباب الثامن عشر : تطوير الـ Charts والتقارير

٥- اضغط زر الـ Add للمحرر واختر Horiz. Bar chart series . اضغط OK . يجب ان ترى بيانات عشوائية معروضة في chart .



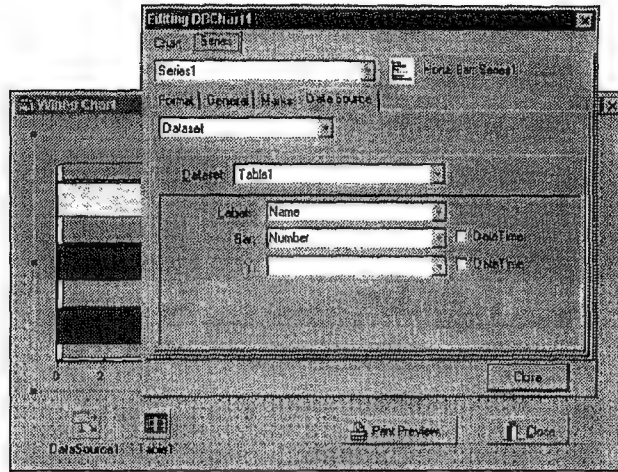
شكل (١٨-١١)؛ عرض وقت التصميم لنافذة الـ WinesChart أيضاً يوضح الـ DataSource1 والـ Table1 التي تربط البرنامج بقاعدة بيانات الـ Wines

٦- لربط أيضاً chart بقاعدة بيانات WINES ، اختر الـ Series page tab ، لمحرر الـ TeeChart في أعلى النافذة . ولان هناك series objects واحدة ، فتكون قد تم اختيارها بالفعل (في برامجك ، اذا استخدمت series objects متعددة ، اختر كل واحد من قائمة اللائحة ثم اعد الخطوة التالية) .

٧- اضغط الـ Data Source tab في محرر الـ TeeChart . افتح قائمة اللائحة تحت الصف الثاني من page tabs واختر مدخل الـ Dataset . بنفس الطريقة ، حدد الـ Dataset edit box بـ Table1 . هذه الخطوات تربط chart بـ Table قاعدة البيانات . لتحديد أى الحقول يتم عرضها ، اختر الـ Label Name edit box و الـ Bar Number . هذه القيم تحدد chart تحسب عدد الزجاجة في مخزن خمور يحمل بطاقة حقل الـ Name لكل زجاجة . يوضح شكل (١٨-١٢) محرر الـ TeeChart مع الـ SeriesData Source page وهي ممتلئة .

٨- اختر محرر الـ TeeChart لترى الخريطة التامة في نافذة الـ form ، موضحة المعلومات المأخوذة من قاعدة بيانات الـ WINES . لا يجب عليك تشغيل البرنامج-لأنك قد حددت خاصية الـ Active لـ Table1 بـ True ، فإن البيانات تكون متواجدة داخل Delphi .

لإنهاء البرنامج، يمكنك تغيير عناوين ال chart واختيار خصائص أخرى .  
 فى الواقع ، هذا سيكون مكاناً جيداً لك لتبدأ واختبار خيارات ال chart المتنوعة .  
 توضح قائمة البرنامج [انظر القائمة (١٨-٤)] كيفية برمجة زر ال Print Preview... كما سبق التوضيح فى هذا الباب .



شكل (١٨-١٢): استخدم محرر ال TeeChart لربط chart series بال Table لقاعدة بيانات، فى هذه الحالة، لقاعدة البيانات WINES

## إنشاء تقارير مع ال QuickReport:

يبدو ان طابعات الحاسب اليوم تعمل بشكل جيد، ولكن هذا ليس دائماً .  
 ليس من زمن بعيد، كانت عملية جعل الطباعة تطبع " أى شئ " تعد إنجازاً كبيراً  
 لكثيرين من مستخدمي الحاسب . ولكن الآن، اصبحت طابعات الليزر تحتل  
 العمل الشاق . ولكن، لكى تجعلها تنطلق كما تريدها مازال امرأ يحتاج الى التجربة  
 والخطأ .

ان ال QuickReport library الخاصة بـ Delphi يمكن ان تبسط متطلبات  
 الطباعة الخاصة بتطبيقك، خاصة اذا كنت تحتاج الى اعداد تقارير مطبوعة من ملف  
 أو بيانات قاعدة بيانات . تأتى مع محرر سهل الاستخدام والذى يبسط تخطيط  
 الصفحة . وكذلك هناك ميزة معاينة الشكل النهائي للطباعة وهى توضح على  
 الشاشة بالضبط كيف ستظهر المطبوعات النهائية، والتي يمكنك استخدامها ايضاً  
 لحفظ تقارير فى ملفات قرص .

**ملحوظة:** ان التوثيق الكامل للـ QuickReport library متوفر في دليل تركيب Delphi، فإننى لا اعرف اسم المسار لهذا الدليل، ولكن فى Delphi 4، فإننى لا اعرف اسم المسار لهذا الدليل، ولكن فى Delphi 3، كان المسار، (Delphi 3\Quickrpt. افتح ملف الـ Qrptxman.doc (x تساوى رقم النسخة) باستخدام متصفح الـ Microsoft Word أو الـ Word . وتعتبر الـ online help الكاملة متاحة ايضاً- اختر أى QuickReport component أو بند آخر واضغط F1 لمزيد من المعلومات .

Note

### البداية مع التقارير:

ان الـ QuickReport يعتبر مولد الـ banded report. باختصار هذا يعنى انك تنشئ تقريراً بواسطة إدخال واحد أو أكثر من الـ band objects فى صفحة خالية، وهو الـ object للـ QuickReport. وفى كل الـ bands، تقوم باسقاط عناصر متنوعة تريدها ان تظهر فى التقرير، مثل العناوين، التاريخ والوقت، ارقام الصفحات، حقول تسجيل قاعدة البيانات، إجماليات ملخصة، وعناصر اخرى. عندما تطبع التقرير، تملأ الـ QuickReport الـ bands بالمعلومات وتنسخها حسب الحاجة لإنتاج منتج تام. على سبيل المثال، فى كل صفحة، تقوم الـ QuickReport بصورة تلقائية بزيادة رقم الصفحة، و تملأ حقول التسجيل بمعلومات مأخوذة من الـ tables قاعدة بيانات.

وعندما تعمل مع الـ QuickReport، سوف تكتشف أن هذه الـ bands غاية فى الذكاء. فى اغلب الحالات، يمكنك تصميم تقارير جيدة الشكل بإنشاء الـ bands قليلة وإسقاط الـ objects عليها. إنك تحتاج الى قليل من البرمجة. وميزة معاينة الشكل النهائى للطباعة السهلة الاستخدام، التى يمكنك استخدامها إما فى نط تصميم الـ Delphi أو فى تطبيق وقت التشغيل، تجعل من السهل رؤية تخطيط التقرير دون إهدار الورق. عندما يكون كل شئ مضبوط كما تريده، اضغط زر الـ Print الخاص بمعاينة الشكل النهائى للطباعة لترسل النتيجة فى طريقها الى طابعتك. يمكنك ايضاً حفظ التقارير فى ملفات قرص ثم تعيد تحميلها فى نافذة معاينة الشكل النهائى للطباعة التابعة للـ QuickReport لىتم طباعتها.

**على القرص المدمج:** لعرض كيفية استخدام ال QuickReport components ، إتبع الخطوات المذكورة فى هذا الفصل والفصول العديدة القادمة . حتى اذا لم يكن لديك طابعة ، يمكنك إتباع هذه التعليمات خطوة بخطوة لتعلم كيفية تصميم تقرير . (والتطبيق التام ، Report1 ، موجود على القرص المدمج فى دليل Source\Report1 . لاستخدام هذا البرنامج ، أو لإنشاءه باتباع الخطوات الموجودة هنا ، يجب ان يكون قد قمت بتركيب ملفات عرض قاعدة بيانات Delphi ، والمعرفة بال Alias name (DBDEMOS) . ابدأ تطبيقاً جديداً ثم اتبع الخطوات التالية :



١- اضغط ال Data Access page tab ، اختر component وقم باسقاط على ال form . اختر ال DataSource واسقطه الى جانب ال Table1 المواضيع بدقة لا تهم- فلا يظهر أى من ال component فى التطبيق التام .

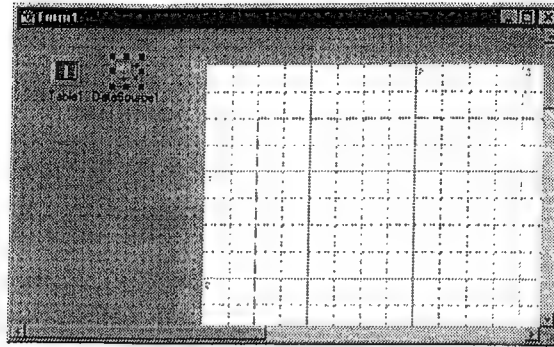
٢- اضغط ال Table1 ، قم اضغط ال Properties tab فى نافذة ال Object Inspector ل Delphi حدد ثلاث خصائص كما يلى : ال DatabaseName بـ DBDEMOS ، ال TableName بـ Parts.db ، وال Active بـ True .

**ملحوظة:** اذا كان ال DBDEMOS غير مذكور فى قائمة اللائحة لخاصية ال DatabaseName لل Table1 ، فإنك تحتاج الى إعادة تركيب Delphi وتبدأ من جديد . أو ، يمكنك اختيار أى قاعدة بيانات وجدول آخرين متوفرين على نظامك ، ثم حدد أى مجالات متاحة مكان تلك المقترحة هنا . انظر الباب السابع عشر لمزيد من المعلومات عن استخدام components قاعدة البيانات .



٣- اضغط ال DataSource1 ، وباستخدام ال Object Inspector ، حدد خاصية ال DataSet لها بـ Table1 .

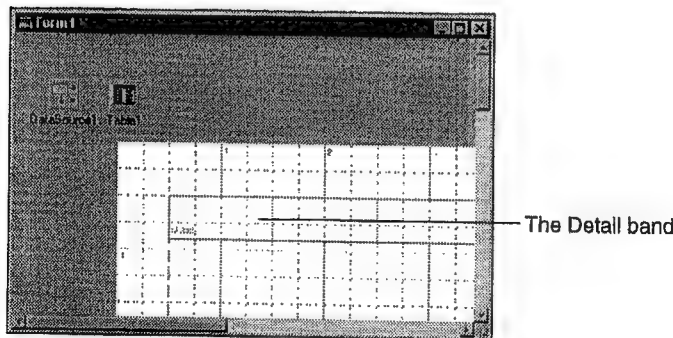
٤- اضغط باب ال QReport page tab ل Delphi على ال VCL palette ، واضغط الايقونة الاولى الى اليسار ، وهى QuickRep . اضغط داخل نافذة ال form لإنشاء شكل تقرير خالى ، يحمل اسم QuickRep1 حسب النظام الافتراضى . يوضح شكل (١٨-١٣) نافذة ال form فى غط التصميم حتى الآن . وال grid وارقام الصف والعمود الخاصة بها والتي تراها على الشاشة هى لتخطيط عناصر التقرير ، ولا تظهر فى التقرير التام .



شكل (١٨-١٣): شكل تقرير خالي مع components قاعدة البيانات

٥- اضغط داخل شكل التقرير الخالي ، ثم في نافذة الـ Object Inspector ، اضغط مرتين علامة الزائد الى اليسار من خاصية الـ Bands. هذا يفتح قائمة بالخصائص الفرعية ، كل واحدة لها تحديد True أو False والذي ، عندما يكون True ، ينشئ band من هذا النوع في شكل التقرير .

٦- حدد الخاصية الفرعية HasDetail بـ True. هذا ينشئ شكل band في التقرير . اذا نظرت من قرب ، يمكنك ان ترى الكلمة الباهتة "Detail" داخل هذا الـ HasDetail. هذا يعرف نوع الـ HasDetail ، ولكن لا يتم طباعته في التقرير التام . ان شكل الـ QuickRep1 مع شكل الـ band موضح في شكل (١٨-١٤) . (انظر على شاشة عن الـ "Detail" داخل الشريط هذا النص يكون غالباً صغيراً جداً بحيث يصعب رؤية في الشكل المطبوع) .



شكل (١٨-١٤): شكل الـ QuickRep1 مع شكل الـ band



٧- تأكد من ان الـ QuickRep مازال مختاراً (فهو مذكور فى أعلى قائمة اللائحة الخاصة بالـ Object Inspector)، حدد خاصية الـ DataSet لها بـ Table1. اصبح الآن شكل التقرير مرتبط بـ components قاعدة البيانات التى يتم اخذ المعلومات منها لإنتاج التقرير التام.

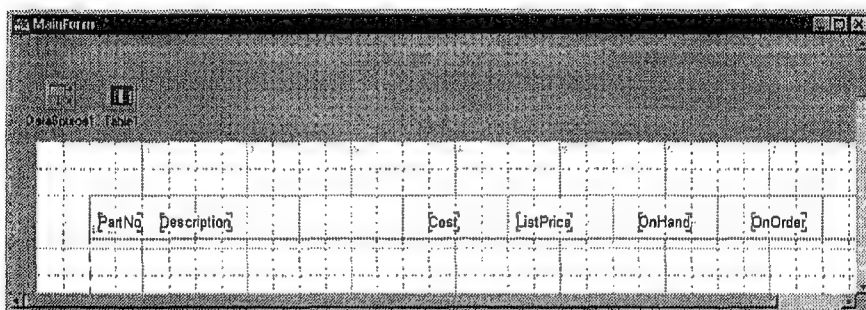
٨- اضغط داخل الـ DetailBand1 لاختياره، ولاحظ ان الـ Object Inspector يعرض اسم شكل الـ band، DetailBand1. يقوم الـ QuickReport بنسخ الـ Detailband فى التقرير التام لطباعة صفوف من المعلومات.

٩- لتوفير تلك المعلومات، قم بإضافة بعض الـ detail objects، على الـ band. ضع زر الـ QRDBText فى الـ QReport palette الخاصة بالـ VCL (يوجد العديد من الايقونات فى هذا الفصل من Delphi - اوقف الفأرة على كل زر وانتظر لحظة لترى اسمه هذا يساعدك على إيجاد الـ component الذى تحتاجه). اضغط الـ QRDBText، ثم اضغط داخل الـ Detail band للتقرير. يمكنك سحب شكل النص الناتج (الذى يحمل الـ labeled QRDBText1) الى أى موضع تريده داخل الـ Detail band - أى مكان الى اليسار فى الـ band.

١٠- تأكد من ان الـ QRDBText1 مختاراً، ثم فى نافذة الـ Object Inspector، حدد إثنين من الخصائص كمايلى: الـ DataSet لها بـ Table1 والـ DataField بـ PartNo. هذه الخطوات تحدد ان العمود الأول فى التقرير يذكر رقم جزء العنصر. لاحظ ان الـ band يوضح اسم الحقل المختار.

١١- اعد الخطوة رقم (٩) لإضافة حقول الـ Table قاعدة بيانات إضافة الى الـ Detail band بالتقرير على سبيل المثال، اختر مرة اخرى الـ QRDBText component فى الـ palette، واضغط فى الـ Detail band لإنشاء الـ QRDBText 2. حدد الـ DataSet الخاصة بهذا الـ object بـ Table1، ولكن هذه المرة، حدد الـ DataField بـ Description اصف الـ QRDBText components إضافية لحقول الـ Cost، والـ ListPrice، والـ OnHand، والـ OnOrder. اعد تحديد حجم نافذة الـ form على حسب الحاجة لعرض مزيد من التقرير المتطور. يوضح شكل (١٨-١٥) الـ Detail band التام.

**فكرة:** اعتماداً على نوع الحقل الذى تقوم بطباعته فى العمود، قد تريد ان تحدد خاصية الـ Alignment له بشئ مختلف . على سبيل المثال، ان عمود الـ Description فى التقرير يجب ان يحدد الـ Alignment بـ taLeftJustify . والحقول الرقمية الاخرى سوف تبدو افضل اذا تم تحديدها بـ taLeftJustify يجب عليك ان تعمل بهذه أو تلك الخصائص، وقد تحتاج ان تعدل مواضع الاعمدة، لتحصل على النتائج النهائية كما تريدها.



شكل (١٨-١٥) الـ Detail band التام. كل الـ QRDBText يمثل  
حقل Table قاعدة بيانات يجب طباعته فى التقرير التام.

١٢- احفظ التطبيق الآن . إنك تستمر فى تطوير هذا التقرير فى الفصول التالية . استخدم الوحدة الافتراضية واسماء المشروع الافتراضية، واحفظهم فى أى دليل مؤقت .

**تحذير:** يعتبر الـ band التقرير object containers . هذا يعنى انه، اذا قمت بحذف الـ band، فإنك أيضاً تحذف الـ objects الموجودة فى الـ band . والأهم من ذلك . التابعة للـ QuickRep بـ False، أو يكون عليك ان تعيد ادخال كل الـ objects التى ادخلتها بعناية فى هذا الـ band . للسلامة، بعد إنشاء كل الـ band وإدخال الـ band فيه، احفظ مشروعك .

نتج الخطوات، اضغط داخل الـ Detail band). اختر امر الـ Preview من القائمة لرؤية التقرير كما تم تجميعه حتى الآن . يوضح شكل (١٨-١٦) النتائج .

**ملحوظة:** بالرغم من أنك تجرب اوامر معاينة الشكل النهائى والطباعة الخاصة بالـ QuickReport داخل Delphi، يرى المستخدمون

لتطبيقك نفس نافذة dialog معاينة الشكل النهائي . مؤخراً فى هذا الباب ، سأشرح كيفية توفير هذه الميزة فى تطبيقاتك التامة .

800	Dive kayak	\$1,358.75	\$3,899.85	24	18
912	Underwater Diver Vehicle	\$504.00	\$1,880.00	5	3
1313	Regulator System	\$117.50	\$250.00	165	216
1314	Second Stage Regulator	\$124.10	\$385.00	98	88
1316	Regulator System	\$119.35	\$341.00	75	70
1320	Second Stage Regulator	\$73.53	\$171.00	37	35
1328	Regulator System	\$154.80	\$430.00	168	100
1330	Alternate Inflation Regulator	\$85.80	\$280.00	47	43
1384	Second Stage Regulator	\$99.90	\$270.00	128	135
1390	First Stage Regulator	\$64.60	\$170.00	146	140
1946	Second Stage Regulator	\$95.79	\$309.00	13	10
1986	Depth/Pressure Gauge Console	\$73.32	\$188.00	25	24
2314	Electronic Console	\$120.80	\$390.00	13	12
2341	Depth/Pressure Gauge	\$48.30	\$105.00	225	225

شكل (١٨-١٦)؛ امرال Preview الخاص بال QuickReport يعرض التقرير كما سيظهر على الصفحة المطبوعة

جرب الازرار الموجودة على صفحة معاينة الشكل النهائي للطباعة . اضغط Zoom لعرض الصفحة كاملة ، اضغط Zoom to 100 لتحصل على تمثيل واضح على الشاشة للتقرير المطبوع (هذا الخيار مفيد لوضع ال objects بعناية قبل الطباعة) . اضغط ازرار الاسهم لترى صفحات التقرير . اضغط Printer setup لتعد خيارات طابعتك ؛ اضغط Print للطباعة (تبدأ الطباعة على الفور- فهى لا تظهر خصائص طباعة ، لذا تأكد من اختيار Printer setup قبل ذلك) . وهناك زرین آخرين وهما save و Open ، يحفظان التقرير الحالى فى ملف قرص ويعيدا تحميل القاریر التى تم حفظها فيما سبق . اضغط زر ال Close لتعود الى صفحة chart form .

## طباعة رؤوس العمود:

حتى الآن، يبدو تقريراً مقبولاً، ولكن الاعمدة غير معرفة، واغلب المعلومات في الإخراج النهائي ستكون بلا معنى [انظر شكل (١٨-١٦)]. ان كل عمود يحتاج الى الـ label، وافضل ما يكون طباعتها باستخدام بنطاً مختلفاً ليكون رأس العمود بارزاً.

ورأس العمود هو مجموعة من الـ QRLabel، والتي تستطيع ان تظهر أى نص تريده. وهذه الـ objects تشبه الـ label الخاصة بـ Delphi، ولكن تم تصميمها لتدخل في الـ QuickRep object bands. ولان، في هذه الحالة، يجب طباعة رأس العمود فقط على قمة كل صفحة فانت تحتاج ان تنشئ band تقرير مختلف ليحمله. اذا وضعت labels في Detail band، فستكرر في كل سطر جديد من المخرجات (قد يكون هذا مفيداً في بعض الاحوال، ولكنه ليس التأثير المطلوب هنا).

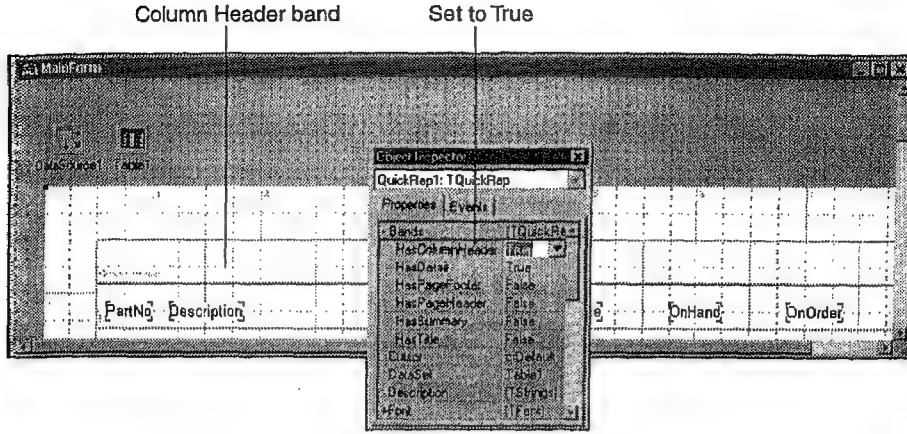
**على القرص المدمج:** اذا كنت مازالت ترى نافذة معاينة إغلاقها لتعود الى Delphi، ثم اتبع الخطوات التالية لإضافة رؤوس عمود الى التقرير (مرة اخرى، تطبيق التقرير التام موجود على القرص المدمج في دليل



: (Source\Report1

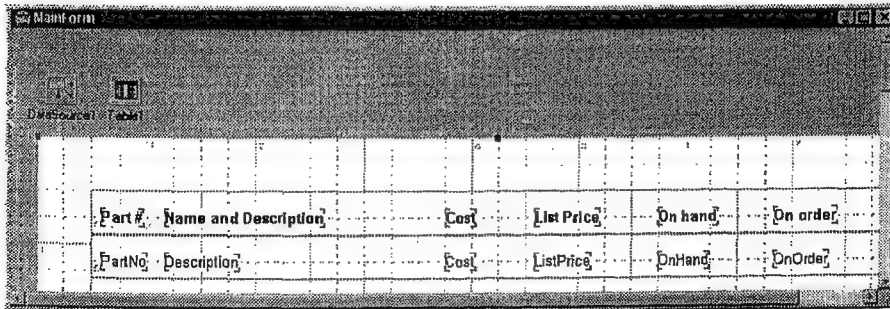
١- اختر شكل QuickRep1 (يجب ان يظهر اسمه في نافذة الـ Object Inspector). اضغط مرتين علامة الزائد الواقعة بعد خاصية الـ Bands، وحدد خاصية الـ HasColumnHeader بـ True. هذا يضيف آخر Band للـ QuickRep1، يحمل الـ label الباهتة ColumnHeader (هذا النص لا يظهر في التقرير التام). ويوضح شكل (١٨-١٧) التقرير مع الـ Column Header band الجديد.

٢- اختر QRLabel من الـ VCL QReport palette، واضغط داخل الـ Column Header band لإدخال label تحمل اسم QRLabel1 حسب النظام الافتراضى. اسحب الـ QRLabel1 الى أعلى العمود الذى يحتاج، ثم استخدم الـ Object Inspector لإدخال كتابة أى نص تريده- لا يجب ان يتناسب مع اسم حقل الرأس. يوضح شكل (١٨-١٨) الـ Column Header band التام مع الـ QRLabel1 أعلى كل عمود.



شكل (١٧-١٨): حدد الـ `HasColumnHeader` بـ `True` لإضافة `Column Header band`، والذي يطبع `labels` عمود على كل صفحة من التقرير التام

٣- من الأفضل دوماً أن تطبع رؤوس الأعمدة باستخدام `fonts` مختلفة . يمكنك اختيار `fonts` لكل `QRLabel` الأخرى باستخدام خصائص الـ `Font` لهم . ولكن، لكي تغير الـ `font` للـ `band` بأكمله، اختر الـ `ColumnHeader band` قائمة لائحة نافذة الـ `Object Inspector` توضح الـ `ColumnHeaderBand1`، ثم اضغط الزر البيضاوي الواقع بعد خاصية الـ `Font`. في `dialog` الاختيار الـ `font` الناتج، اختر `Bold`. يمكنك أيضاً أن تختار ألوان المخرجات، أسماء عائلة الـ `font`، أحجام النقاط، وأن تصنع التغييرات الأخرى في الـ `font` المختار وخصائصه .



شكل (١٨-١٨): الـ `Column Header band` التامة لكل عمود مع الـ `QRLabel`

## الباب الثامن عشر: تطوير الـ Charts والتقارير

٤- عندما تنتهى من وضع labeling الاعمدة، اضغط داخل الـ QuickRep1 (لا تضغط داخل أى من الـ two bands)، واختر امر الـ Preview من القائمة المحلية. يمكنك عندئذ معاينة الشكل النهائى وطباعة التقرير بكل اعمدة حاملة لـ labeled.

٥- احفظ التطبيق الآن. إنك تستمر فى تطوير هذا التقرير فى الفصول القادمة.

**فكرة:** تأخذ كل الـ objects خصائص الـ font من الحاوية الأم لهم. Tip  
يستخدم الـ bands الـ font للتقرير، والـ objects الموجودة فى الـ bands تستخدم band fonts، الى آخر ذلك. لتغيير الـ font للتقرير بأكمله، اختر الـ QuickRep وحدد خاصية الـ Font له. يمكنك عندئذ تعديل خصائص الـ Font للـ bands المنفردة والـ objects المحتواه فيها.

### طباعة معلومات النظام:

إنك غالباً تريد ان تضيف معلومات نظام لصفحات التقرير. وهذه المعلومات قد تعرض عنوان تقرير، التاريخ والوقت الحاليين، ورقم صفحة. كما فعلت مع الـ labels العمود والحقول الخاصة بالتقرير، لإضافة معلومات نظام، فإنك تنشئ نوع آخر من الـ band تسقط فيه QuickReport.

اغلق نافذة معاينة الشكل النهائى الخاصة بالـ QuickReport اذا كانت مفتوحة، وإرجع الى Delphi. ثم، إتبع الخطوات التالية لإضافة عنوان، تاريخ ووقت، ورقم صفحة للتقرير:

١- اختر QuickRep1، اضغط مرتين علامة الزائد الواقعة الى اليسار من خاصية الـ Bands، وحدد خاصية الـ HasColumnHeader بـ True. هذا يضيف آخر Band ثالث الى التقرير، كما هو الحال مع الـ band الأخرى، يظهر النص الـ Page Header فقط فى وقت التصميم، وليس فى المخرجات المطبوعة. (قد يكون عليك ان تنظر الى شاشتك لترى هذا النص).

٢- ان الـ objects الموجودة فى الـ Page Header band تكون غالباً QRLabel أو QRSysData objects. لإضافة عنوان الى أعلى الصفحة، اختر

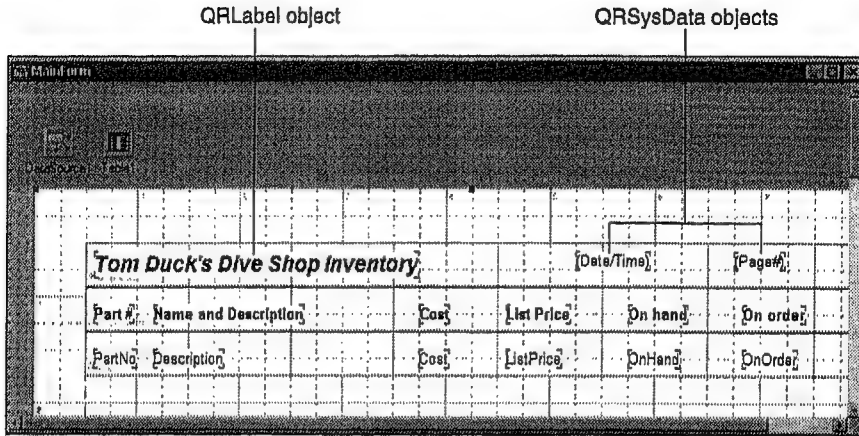
[illegible]

٣- اختر QRSysData، وقم بإسقاط اثنين منه في الPage Header band. كل واحد من هذين ال two components يمكن ان يعرض انواع مختلفة من معلومات النظام الثابتة. اختر كل object، وحدد خاصية الData بنوع المعلومات الذى تريده- فى هذه الحالة، لقد حددت ال object الاول به qrsDateTime والثانى به qrs PageNumber. ضع ال two components اينما تريد. يوضح شكل (١٨-٢٠) التقرير مع الPage Header band التام.

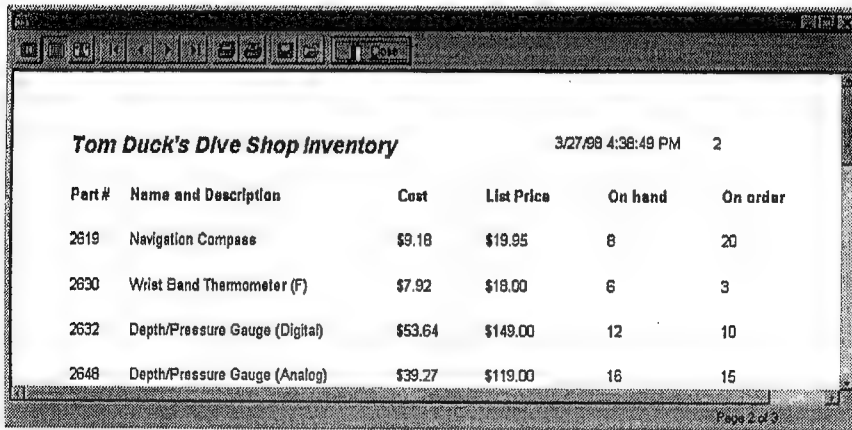
كما فعلت في السابق، اضغط ميمناً داخل الـ QuickRep1 (وليس داخل الـ bans)، وثم اختر امر الـ Preview. كما يوضح شكل (١٨-٢١)، يبدأ التقرير

## الباب الثامن عشر : تطوير الـ Charts والتقارير

التمام فى الظهور بشكل طيب ، ولكن بعض البنود يمكن اضافتها لتنمية بعض الشئ .



شكل (٢٠-١٨) : الـ Page Header band الكامل مع عنوان الـ QRLabel و QRsysData objects موضحة التاريخ والوقت، ورقم الصفحة



شكل (٢١-١٨) : التقرير الذى أوشك على الانتهاء مع عنوان الصفحة، تاريخ ووقت، رقم صفحة، رؤوس اعمدة، ومعلومات قاعدة بيانات

**فكرة:** بدلاً من استخدام الـ Page Header band لطباعة عنوان التقرير ، يمكنك تحديد الخاصية الفرعية HasTitle للـ Bands بـ True . يمكنك عندئذ إضافة title label للـ Title Band-هذا الـ Band يتم طباعة على الصفحة الأولى فقط .





طباعة معلومات فى اسفل كل صفحة، حدد الخاصية الفرعية HasFooter للـ Bands التابعة للـ QuickRep بـ True. استخدام band الناتج تماماً مثل الـ Header Band الموضح فى هذا الفصل. بالتأكيد، يتم طباعة الـ object التى تدخلها فى هذا الـ band فى اسفل كل صفحة بدلاً من اعلاها.

### تلخيص الاعمدة:

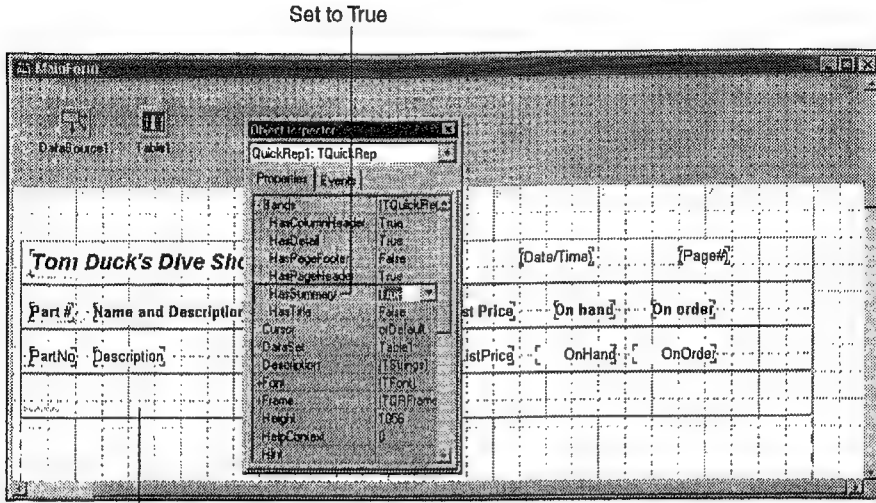
لجمع القيم فى عمود رقمى وتوفير معلومات ملخصة اخرى، فإنك تحتاج الى band واحد اضافى واثنين من الـ components لهذا التقرير على سبيل المثال، التقرير المتطور [راجع شكل (١٨-٢١)] قد يجمع القيم فى اعمدة On Hand أو On Order. ولأغراض الايضاح، قمت ايضاً باظهار مجموع لعمود الـ List Price، بالرغم من ان هذا لا يبدو مقيداً فى هذه الحالة. ولكن، القيمة الإجمالية لمخزن المحل ستكون مفيد للغاية، لذا صممت هذا فى التقرير النهائى.

لإضافة band تلخيص الى التقرير، اغلق نافذة dialog معاينة الشكل النهائى للـ QuickReport اذا كانت مفتوحة، وراجع الى Delphi. ثم اتبع الخطوات الآتية:

١- اختر QuickRep1، وفى الـ Object Inspector، اضغط مرتين خاصية الـ Bands، وحدد خاصية الـ HasSummary بـ True. يوضح شكل (١٨-٢٢) شكل التقرير، النامى مع band الجديد. كما هو الحال مع الـ bands الأخرى، يظهر النص الـ Summary فى Delphi فقط، وليس فى المخرجات المطبوعة.

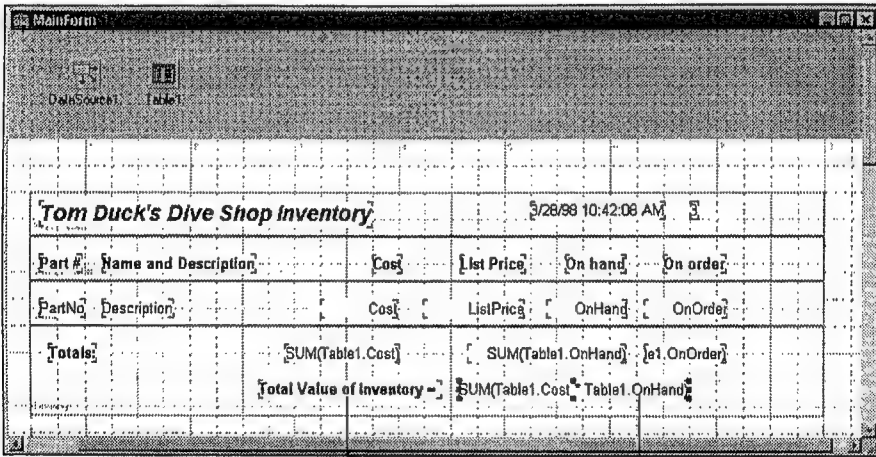
٢- يظهر الـ Summary band فى اسفل chart بعد السطر الاخير من البيانات. يمكنك اضافة QuickReport متنوعة الى هذا الـ band. فى هذا الحالة، نصف تحتاج الى نوعين QRLabel objects لتعريف قيم التلخيص وQRExpr objects لأداء الحسابات. (تحمل ايقونة الـ QRExpr لـ  $E = mc^2$  labeled). صنع اثنين من الـ QRLabel objects على الـ Summary band واربعة من الـ QRExpr objects. اضغط واسحب الخط الخارجى للـ Summary band لتكبيره، ورتب الـ objects كما هو موضح فى شكل (١٨-٢٣). حدد خصائص الـ QRLabel Caption كما هو موضح فى الشكل، واستخدم خاصية الـ Font لاختيار لون للمخرجات.

## الباب الثامن عشر : تطوير الـ Charts والتقارير



Summary band

شكل (١٨-٢٢) : اضع Summary band الى التقرير لطباعة  
اجماليات اعمدة ومعلومات ملخصة اخرى فى اسفل التقرير



QLabel object

QExpr object

شكل (١٨-٢٣) : الـ Summary band التام مع الـ QLabel objects والـ QExpr

٣- لحساب قيم الـ Summary ، اختر كل QExpr ، وحدد كل خاصية Expression كما يلى . ويتم عرضها فى الـ form المصممة وفى التقرير التام ، يتم استبدالها بالقيم المحسوبة . لاحظ انه للإشارة الى حقل فى Table قاعدة البيانات ،

يشير التعبير الى الTable1 الخاص الform باستخدام نظام النقاط زائد اسم الحقل (Cost ، OnHand ، الى آخره) . وهذه ليست برمجة Pascal- يتم تقييم نفس التعبير بواسطة QuickReport :

Sum(Table1.Cost)

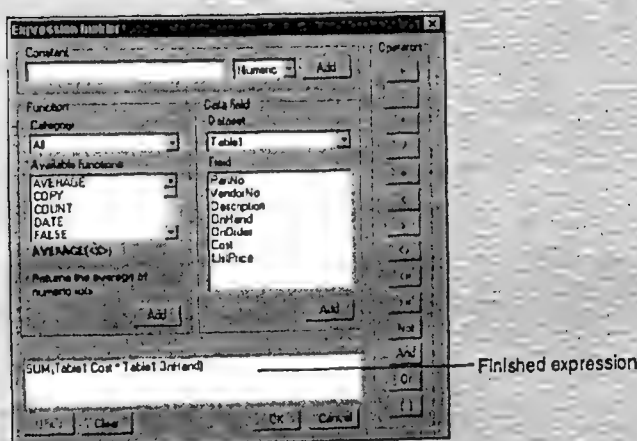
Sum(Table1.OnHand)

Sum(Table1.OnOrder)

**Sum(Table1.Cost \* Table1.OnHand)**

**QuickReport Expression Builder** منشیء تعبیر الـ

بدلاً من كتابة التعبيرات في خصائص الـ QRExpr Expression، يمكنك أن تضغط الزر البيضوي لهذه الخاصية لفتح الـ QuickReport Expression Bulder (أو منشئ تعبير الـ QuickReport). انظر شكل (١٨-٢٤). استخدم المحرر، اختر Function، واضغط Add في الـ panel اليسرى. على سبيل المثال، اختر SUM. بعد ذلك، اختر Dataset object (هناك واحداً فقط في هذا المثال، وهو Table1)، واختر Field مثل Cost. اضغط Add في الـ panel اليسرى لتضيف هذا إلى التعبير. اختياريًا، اضغط Operator في الـ panel اليمنى البعيدة، ثم اختر Field آخر مثل OnHand. أخيراً، اضغط OK لإتمام التعبير، الذي يكون مؤشراً في النافذة السفلى. عندما ترضى عن هذا، اضغط OK مرة أخرى لتدخل التعبير في خاصية الـ QRExpr Expression وتعود إلى تصميم الـ form.



شكل (١٨-٢٤): بدلاً من إدخال خصائص الـ QRExpr Expression بالكتابة، يمكنك استخدام الـ QuickReport Expression Builder لإنشاء عبارات

## الباب الثامن عشر : تطوير الـ Charts والتقارير

٤- بالإضافة الى تحديد خصائص الـ QRExpr Expression ، فى الغالب تريد ان تدخل خاصية Mask لتصوير القيمة المطبوعة . قد تريد ايضاً ان تفعل نفس الشئ للـ QRDBText فى الـ Detail band-ان الـ Masks تعطيك تحكماً أفضل فى محاذاة الاعمدة ، وتمكنك من تحديد عدد الاماكن العشرية فى القيم الرقمية . فيما يلى خصائص الـ mask المستخدمة فى التقرير ونماذج من نتائجها :

#,###.00	2,609.20
\$#,###,###.00	\$24,891.05
####	43

### Masks

فى الـ mask ، علامة (#) تمثل أى خانة رقمية، ولكن يتم طباعتها فقط اذا كانت القيمة لها رقم فى هذا الموضع. والصفر يمثل أى رقم، ويتم طباعته دائماً حتى اذا لم يكن بالقيمة رقم فى هذا الموضع. والفصلات تشير الى الفاصل الذى تريد استخدامة فى القيم الكبيرة.

ان نوع الـ mask يختلف اعتماداً على نوع حقل البيانات. ان الـ TDateField ، والـ TDateTimeField ، والـ TTimeField objects يتم تهيئتها باستدعاء الـ DateTimeToStr function الخاصة بـ Delphi. ولهذا الـ components ، اذا لم يتم ادخال خاصية الـ Mask ، تتطابق المخرجات مع المواصفات الموجودة فى ملف بدء الـ Windows ، وهو الـ Win.ini ، فى القطاع [الدولى].

ان الـ TBCDField ، والـ TCurrencyField ، والـ TFloatField يتم تحديد الـ format لها باستدعاء الـ FloatToTextFmt function التابعة لـ Delphi. لهذه الـ components ، اذا لم يتم ادخال Mask أو Display form ، يتم تحديد الـ format القيمة باستخدام خاصية الـ Currency للمجال.

٥- بعد تحديد خصائص الـ Expression والـ Mask فى الـ QRExpr الخاصة بك ، اضغط ميمناً الـ QuickRep1 (لاتضغط داخل الـ band) ، واختر امر الـ Preview من القائمة المحلية . يوضح شكل (١٨-٢٥) معلومات التلخيص للتقرير التام مع إجمالى العمود وتقرير عن قيمة المخزن الإجمالى للمحل .

٦- احفظ التطبيق الآن . سوف تستمر فى تطوير هذا التقرير فى الفصول

القادمة .

## Sorting بيانات التقرير:

ان الـ sort تعتبر أحد عمليات قاعدة بيانات، وليس QuickReport function. لترتيب التقرير، فإنك تختار index لـ table قاعدة البيانات اذا لم يوجد index للحقل، استخدم الـ Database Desktop لإنشاء واحداً.

**Tip فكرة:** ولكنك قد تتساءل، ماذا لو كنت لاتستطيع ان تنشئ index جديداً لان Database Desktop قاعدة بيانات للقراءة فقط، أو انها متاحة فقط عبر Server بعيد؟ في مثل هذه الحالات، اقترح عليك ان تطبع التقرير على ملف نص قرص، ثم قم بترتيب سطور النص. اذا كان لديك متطلبات ترتيب خاصة، يمكنك قراءة الـ code لتقوم بالترتيب، أو اذا كنت تريد ترتيباً أبجدياً أو رقمياً، فيمكنك ان تفعل كما فعلت انا. قم بتحميل النص في برنامج معالجة كلمات خاص بك واستخدم امر الـ Sort للبرنامج. يمكنك عندئذ ان تطبع ملف النص المرتب. وهذا عادة يكون اسهل بكثير من تجربة ان ترتب قاعدة بيانات حقيقية.

لترتيب تقرير المحل الخاص بنا بوصف العنصر، اغلق نافذة مراجعة الشكل النهائي للـ QuickReport اذا كانت مفتوحة وعد الى Delphi. اتبع هذه الخطوات:

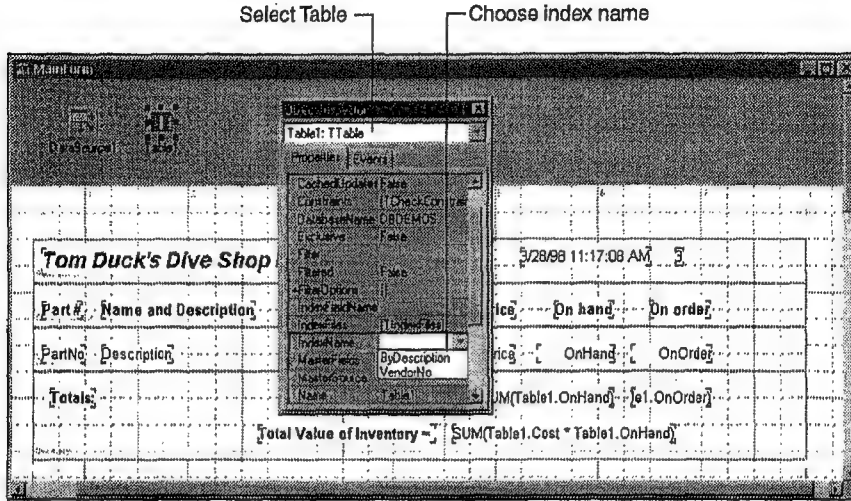
- ١- اختر Table. في الـ Object Inspector (اضغط F11 اذا لم تكن مرئية)، اضغط السهم المشير الى اسفل الواقع يعد خاصية الـ IndexName. هذا يعرض indexes الحقل المتاحة. اختر ByDescription [انظر شكل (١٨-٢٥)].
- ٢- اضغط ميمناً الـ QuickRep1 (ولكن ليس داخل الـ band) واختر أمر الـ Preview لترى وتطبع التقرير التام.
- ٣- اضغط التطبيق الآن. إنك تحتاجه مرة أخيرة في الفصل التالي.

## طباعة التقارير وقت التشغيل:

حتى الآن، لقد قمت بتطوير، معاينة الشكل النهائي وطباعة تقارير من داخل Delphi تماماً. اذا كنت تحتاج الى إنشاء تقارير قاعدة بيانات لنفسك، فهذا مقبول تماماً- فلا يجب عليك أن تكتب أى code أو تتم التطبيق. ولكن، قد تحتاج أن توفر

## الباب الثامن عشر : تطوير الـ Charts والتقارير

إمكانات توليد وتقرير للمستخدمين النهائيين لتطبيقك ، وتعطيهم كل نفس مميزات معاينة الشكل النهائي والطباعة المقدمة في هذا الباب .



شكل (١٨-٢٥)، الترتيب هو عملية قاعدة البيانات وليس التقرير. لترتيب تقرير، اختر الـ Table واختار الـ index المذكور في خاصية الـ IndexName

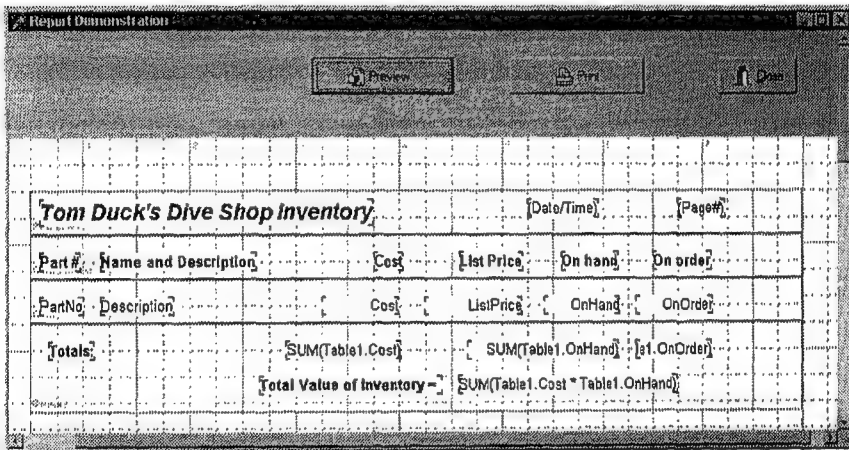
**على القرص المدمج:** اتبع الخطوات التالية لإتمام تطبيق الـ Report1 ولإضافة Buttons لمعاينة الشكل النهائي وطباعة التقرير التام. يوجد البرنامج التام على القرص المدمج في دليل Source\Report1. يمكنك تحميل ملف مشروع الـ Report1.dpr في Delphi، أو إذا كنت متابعاً، أغلق نافذة معاينة الشكل النهائي للـ QuickReport إذا كانت مفتوحة وإرجع إلى Delphi، واتباع هذه الخطوات:

١- أضف إثنين من BitBtn من الـ Additional VCL إلى الـ form، وحدد خصائص الـ Caption لها بـ Preview... ويمكنك إضافة Print BitBtn ثالث وتحديد خاصية الـ Kind له بـ bkClose لتوفر طريقة سريعة للمستخدمين للخروج من التطبيق).

٢- اختياريًا، قم بتحميل bitmaps في خصائص الـ Glyph لكل زر. لقد استخدمت صور الـ Report.bmp والـ Print.bmp المتوفرة مع Delphi في دليل التركيب، Images\Buttons.

٣- اضغط مرتين الـ BitBtn1 لإنشاءOnClick لزر الـ Preview... . إملأ  
الـ procedure باستخدام القائمة (١٨-٥) كمرشد لك . اضغط مرتين الـ BitBtn2  
لإنشاء OnClick ثانياً لزر الـ Print . إملأ هذا الـ procedure أيضاً باستخدام  
القائمة (١٨-٥) كمرشد لك .

٤- اضغط F9 للـ compile وتشغيل البرنامج . يوضح شكل (١٨-٢٦)  
العرض التام للبرنامج . اضغط زر الـ Preview... لإظهار نافذة الـ Preview للـ  
QuickReport . اضغط Print لطباعة التقرير التام- تبدأ الطباعة بمجرد أن تضغط  
هذا الزر . إذا جربت هذا ، فإنك ترى progress indicator على الشاشة [شكل  
(١٨-٢٧) أثناء إنشاء المطبوعات وإرسالها إلى الطابعة . إنك ترى أيضاً ومضة  
بيانات في الـ components المختلفة للتقرير .



شكل (١٨-٢٦): العرض التام للتطبيق الـ Report1 مع أزرار الـ Preview... والـ Print

القائمة (١٨-٥): Report1Main.pas (الـ BitBtn OnClick)  
{ Respond to click of the Preview... button }  
procedure TMainForm.BitBtn1Click(Sender: TObject);  
begin  
    QuickRep1.Preview;  
end;

## الباب الثامن عشر : تطوير الـ Charts والتقارير

```
{ Respond to click of the Print button }  
procedure TMainForm.BitBtn2Click(Sender: TObject);  
begin  
    QuickRep1.Print;  
end;
```



شكل (١٨-٢٧): أثناء الطباعة، يرى  
المستخدمون الـ progress indicator

### افكار للمستخدم الخبير

- أثناء استخدام الـ TeeChart editor ، اضغط الـ Chart والـ General page tabs ، ثم اضغط زر الـ Print Preview... لمعاينة الشكل النهائي وطباعة الـ chart أو ، يمكنك ان تضغط Export لتكتب الـ charts للملفات الحجم الكبير بالقرص أو ملفات الـ bitmap وهذه الازرار اسهل بكثير في الاستخدام اثناء تصميم chart عن الخروج من محرر الـ TeeChart والضغط يميناً داخل الـ chart لمعاينة شكلها النهائي .
- لإضافة chart axis labels ثابتة ، كتلك المستخدمة في تطبيق الـ MayTemp لهذا الباب ، افتح محرر الـ TeeChart ، اختر الـ Chart page tab ، اضغط الصفحة الفرعية Axis . داخل هذه الصفحة ، هناك series اخرى من page tabs . اختر Scales وابطل الـ Automatic check box . بعد ذلك اضغط كل واحد من زرى الـ Change... وادخل الحد الأدنى والحد الأعلى من القيم لتظهر على المحور المختار على اليسار . (تأكد من ان الـ check box الظاهر تم تشغيله ، أو ان تغييراتك لن تظهر على الـ chart التام) . اذا اردت ان تحدد الحد الأدنى أو الحد الأقصى فقط ، اضغط زر الـ Change للقيمة ، وقم بتشغيل الـ Automatic check box للقيمة الأخرى .



- إذا جربت الفكرة السابقة، لا تفحص الـ Automatic check box مرة أخرى، أو سيكون عليك أن تبطله وتعيد إدخال قيم قياس الحد الأدنى والحد الأقصى الخاصة بك.
- لمنع objects التقرير من أن تنطبع فوق بعضها، حدد خصائص الـ AutoSize لها بـ True. كذلك اضغط واسحب الـ objects الفردية (QRLabel، QRExpr، QRDBText، والآخرى) لتحديد حجمها. استخدم خاصية الـ Mask لتصور formats المخرجات.
- من الممكن أن تحصل على Report object ممتد لتحصل على الـ band الخاص به ممتد رأسياً إذا لم يكن هناك مساحة كافية لطباعة نص لـ object. لتفعل هذا، حدد الـ AutoStretch بـ True. ولكن، يجب أن يكون component واحد فقط لكل band هو الذى له هذه الخاصية محددة بـ True- إذا تم تحديد أكثر من object واحد للامتداد بصورة تلقائية، ستكون النتيجة النهائية غير متوقعة (وغير مقروءة طبعاً).
- ان التطبيق Report1 فى هذا الباب له اوامر Preview... و Print ولكن، لان نافذة معاينة الشكل النهائي للـ QuickReport لها زر الـ Print الخاص بها، قد لا يكون ضرورياً ان تضيف امر الـ Print لتطبيقاتك- ان استدعاء الـ QuickRep Preview method يوفر كل خدمات الرؤية والطباعة.
- لطباعة تقارير فى الخلفية، فبدلاً من استدعاء الـ QuickRep Print method، استدع PrintBackground. هذا يمكن المستخدمين من الاستمرار فى استخدام البرنامج أثناء طباعة التقارير الطويلة. يجب أن يكون لديك نسخة ذات 32-bit من Delphi (Version 2 أو ما بعدها) لتستخدم الـ PrintBackground.
- لإضافة chart الى تقرير مطبوع، اختر الـ QRChart من الـ QReport VCL، وأدخل هذا الـ object فى band تقرير. بالرغم من أن الـ QRChart موجود على الـ QRChart، فإنه TeeChart، وهو موثق فى ملفات الـ TeeChart online.

## المشروعات التي يمكنك تجربتها

١٨-١ : قم بتعديل مشروع الـ MayTemp لتحميل ملفات أخرى عن بيانات درجة الحرارة . على سبيل المثال ، قد يستخدم برنامجك أمر الـ FileOpen ، أو زراً ، لعرض dialog ملف مفتوح . اجعل المستخدمين قادرين على فتح ملف درجة حرارة وأقرأ البيانات الموجودة في chart السطر للبرنامج .

١٨-٢ : أضف أمر قائمة أو زر print-preview لواحد من الـ chart النموذجية بهذا الباب وهي MemInfo أو MayTemp .

١٨-٣ : أضف أمر الى التطبيق WinesChart لضغط الـ chart في bitmap . اعرض dialog حفظ ملف لتمكين المستخدم من اختيار دليل واسم ملف للمخرجات .

١٨-٤ : قم بإنشاء تقارير مطبوعة للتطبيقات الموجودة بهذا الباب وهي MayTemp و MemInfo و WinesChart .

## ملخص:

- توفر الـ TeeChart library إمكانات واسعة لصنع الـ charting . تتوفر نسخة كاملة للـ TeeChart library مع Delphi . والنسخة المتخصصة مع الـ source code الكاملة والـ components الإضافية متاحة لدى موزع المنتج ؛ وهو teeMach .
- اختر chart series لإنشاء أنواع مختلفة من الـ pie charts و bar charts للـ chart object أن يكون له series objects واحد أو أكثر ، والذي قد يكون من نفس النوع أو مختلف .
- اضغط يميناً داخل الـ Chart واختر الـ Print Preview... لطباعة ومعاينة للـ charts داخل Delphi .
- لمعاينة الشكل النهائي للـ chart في وقت التشغيل ، والذي يعطى المستخدمين أيضاً القدرة على طباعة الـ chart ، أضف الـ TeePrevi لتقرير الـ

uses للـ unit ، واستدع الـ ChartPreview procedure كما تم التوضيح  
فى التطبيق WinesChart لهذا الباب . (يوجد هذا التطبيق على القرص  
الدمج فى دليل Source\WinesChart).

• توفر الـ QuickReport library إمكانات توليد واسعة للتقرير . تتوفر  
نسخة كاملة من الـ QuickReport library مع Delphi . تتوفر نسخة متخصصة  
مع الـ source code التامة و components إضافية مع موزع المنتج . وهو  
QuSoft .

• يعتبر الـ QuickReport «مولد banded report» لإنشاء تقرير ، قم  
بإضافة QuickRep object على form . استخدم خاصية الـ Bands لهذا الـ  
objects لإنشاء Bands . قم بإضافة objects فردية على هذه لبرمجة التقرير . لا  
تحتاج الا لقليل من البرمجة لإنشاء حتى اطول انواع التقارير وأكثرها توسعاً .

فى الجزء التالى والاخير من هذا الكتاب ، سوف تعرف موضوعات متقدمة  
فى Delphi وبرمجة الـ Object Pascal ، بما فى ذلك الـ exception وإنشاء الـ  
component وتلك الموضوعات مثل كتابة الـ DLLs ، استخدام ادوات خط  
اوامر Delphi ، كتابة event handlers على مستوى التطبيق ، كتابة event  
handlers على مستوى التطبيق ، ومايزيد عن ذلك .

\*\*\*\*\*

## الجزء الرابع تقنيات متقدمة

### محتويات هذا الجزء:

- الباب التاسع عشر: التعامل مع exceptions.
- الباب العشرون: انشاء Custom Components.
- الباب الحادى والعشرون: تطوير مهارات الـ Delphi الخاصة بك.

بعد برمجة التطبيق ، لا يحاول الكثير من المستخدمين التعرف على المزيد من الاشياء . ولكن اغلب المطورين الناجحين يأخذون بعض الوقت لإضافة اللمسات النهائية التى تستطيع ان تحول التطبيق الجيد الى عمل رائع من البرمجيات .

إن الابواب الموجودة فى الجزء الرابع تتناول تقنيات برمجة Delphi المتقدمة . سوف تجد تفسيرات عن كيفية التعامل مع الاخطاء عن طريق ال exception و عن كيفية إنشاء Custom Components و ActiveX control ، وحول كيفية تطوير مهارات Delphi الخاصة بك مع افكار وتقنيات برمجة ربما فى ذلك مقدمة عن كتابة تطبيقات الانترنت .



# الباب التاسع عشر

## التعامل مع ال exceptions

### محتويات هذا الباب:

- معلومات حول ال exceptions.
- التعامل مع ال exceptions.
- إنشاء ال exception classes.
- تقنيات أخرى لل exceptions.

ان التعامل مع الاخطاء هو أحد الضروريات السيئة في حياة مطور البرمجيات . فلا مهرب من هذه المهمة - اذا فشلت تطبيقاتك نتيجة لظرف خاطئ ، فسيكون برنامجك منتشر مثل الفراء في المناطق الاستوائية اثناء موجه حارة . يتعامل التطبيق القوي مع كل الظروف الممكنة والتي قد تجعل البرنامج يتوقف أو ينتج نتائج غير صحيحة .

ومثل هذه الظروف تسمى exceptions والتعامل معها هو موضوع هذا الباب . يوفر ال Object Pascal نظام يعمل مع ال exception غاية في التعقيد ، وكل تطبيق Delphi له exception handlers حسب النظام الافتراضي له يمكن ان يعرضوا رسائل الخطأ ويمنعوا البرنامج من التوقف الغير متوقع اذا حدث exception . وبدمج exception-handling إضافية ، فيمكنك ان تحميك استجابات تطبيقك ضد الظروف الغير متوقعة ، وعندما تحدث مشكلة ، تقوم باعمال المسح اللازمة مثل تحرير الذاكرة المخصصة ، حفظ البيانات الاساسية ، واغلاق الملفات .

## معلومات حول الـ Exceptions:

يقوم المطورون ذوى الخبرة بدمج معالجة الاخطاء فى مشروعاتهم اثناء دورة التطوير . ان معالجة الـ exception فى الـ Object Pascal تعتبر تقنية مختصة بالـ object والتي تقلل كتابة هذا الـ code ان الـ exception تسهل تطوير الـ code و التعامل مع الخطأ معاً، وكما ستعرف فى هذا الباب ان الـ exception handlers لا يحاولون الوقوف فى طريق حسابات البرنامج كما تفعل التقنيات الاخرى . اذا كنت تأمل فى الـ methods افضل للتحكم فى الاخطاء من عبارات الـ if المتداخلة ، و الـ Boolean flags ، وقيم ادخال الـ function الخاصة ، فإن التعامل مع الـ exception هو طريقك .

### أين تنشأ الـ exceptions ؟

يمكن ان تأتى الـ Exceptions من مصادر مختلفة على سبيل المثال ، يستطيع برنامجك ان يولد الـ exception للظروف الغير عادية . تستطيع Delphi components ان تولد exceptions للعديد من الـ events مثل تعيين قيمة خارج النطاق لخاصية ما ، أو محاولة فهرسة عنصر array غير موجود .

ان الـ functions و الـ procedures الـ Run-time library يمكن ايضاً ان تنتج exceptions وان تنفيذ تعبير رياضى عملية غير مسموح بها مثل القسمة على صفر يضمن ظهور الـ exception و الـ methods الاخرى للعمليات المنتجة لـ exception تشمل الاشارة لـ nil pointer ، تخصيص المزيد من الذاكرة أكبر من أكبر block خالى متاح ، وتنفيذ تعبير type-cast غير مسموح به .

اذن ، كل عبارة فى برنامج قد تؤدي الى الـ exception . ولكن ، بعض العبارات تعتبر آمنه نسبياً ، وجزء كبير من مهمتك فى كتابة code قوية هو تحديد أى العبارات تحتاج الى حماية وأيهما لا يسبب ضرراً . وغاذاً هذا الباب تكشف الحالات النموذجية التى يمكن ان تساعدك على اتخاذ هذه القرارات لبرنامجك .

ولكن ، هذا لا يعنى انه يتعين عليك ان تكتب عبارات للتعامل مع كل exception ممكن . ان معالجة الـ exception العامة لـ Delphi ، والتي تم اضافتها لكل تطبيق ، تتولى بصورة تلقائية معالجة الـ exception بعرض الـ dialog box مع وصف للمشكلة وموضع العبارة الخاطئة وهذه الاستجابة الافتراضية حسب النظام

## الباب التاسع عشر : التعامل مع ال exceptions

تعتبر كافية للبرامج الاختبارية والامثلة كاغلب تلك الموجودة فى هذا الكتاب ، ولكنك تحتاج ان تكتب exception handlers للتعامل مع متطلبات برنامجك الفريدة .

بعض انواع العمليات التى يمكن ان ينتج عنها exception تشمل مايلى :

- التعامل مع ملف .
- تخصيص أماكن للذاكرة .
- Windows resources .
- ال Objects و ال forms التى تنشأها فى وقت التشغيل .
- التداخل بين نظام التشغيل ومكونات الأجهزة .

### :Exceptional keywords

يوفر ال Object Pascal العديد من الكلمات الاساسية الخاصة بإنشاء والتعامل مع ال exceptions . وهذه الكلمات الاساسية هى except ، try ، ، on-do-else ، finally ، raise ، at . إنك ستقابلهم جميعاً فى هذا الباب .  
لا تحاول استخدام هذه الكلمات لاغراض اخرى .

**ملحوظة:** ان ال Turbo Pascal compilers و ال Borland الأوائل لا يدركوا كلمات ال exception الاساسية لل Object Pascal . ان التعامل مع Exception تقنية جديدة فى ال Object Pascal compiler الخاص بـ Delphi .



### بعض المصطلحات الجيدة:

فى الفقرات السابقة ، استخدمت بعض المصطلحات الجديدة مثل exception و handler دون توضيح معانيها فيما يلى بعض التعريفات لهذه المصطلحات وغيرها من العبارات التى تحتاج الى معرفتها قبل استخدام ال exception فى تطبيقك :

● **Exception** : هو ظرف غير عادى أو غير متوقع يعترض المسار الطبيعى للبرنامج .



● **Raising an exception**: هو الاشارة الى ظرف غير عادى أو غير متوقع. ان ال Component methods، وال run-time library، اخطاء الاجهزة-حتى تعيين قيمة غير مسوح به لخاصية Component- يمكن ان تحدث exception. ويستطيع تطبيقك ايضاً ان يحدث exception عند اكتشاف سبب يتطلب تعاملأ خاصأ. استخدم الكلمة الاساسية raise لتحداث exception.

● **Exception handler**: هو code يحل المشكلة الذى أحدثت exception. وهذا ال code يجب ان يعيد النظام الى حالة مستقرة حتى يستطيع البرنامج ان يستمر فى التشغيل بشكل طبيعى.

● **Exception instance**: هو object من class تنحدر غالبأ من ال exception class ومعروفة فى ال SysUtils unit. وتحتوى حالة ال exception على معلومات تصف طبيعة الظرف الاستثنائى. ان حدوث exception يخصص ذاكرة لحالة ال exception. وتحرر ال handle exceptions بصورة تلقائية الذاكرة المخصصة.

● **Try block**: هو عبارة واحدة أو أكثر، تسبقها الكلمة الاساسية try، تريد ان تعالج exception فيها أو تريد ان تحرر resource مثل تخصيص الذاكرة منها.

● **Except block**: هو عبارة واحدة أو أكثر تعالج exception الذى حدث بواسطة عبارات فى ال try block السابق. تبدأ ال Except blocks بالكلمة الاساسية except، وقد تحتوى اختيارياً على عبارات on-do-else. يجب ان يتبع ال except block ال try block مباشرة.

● **Finally block**: هو عبارة واحدة أو أكثر يجب ان تنفذ لتحرير الذاكرة المخصصة، لاغلاق ملفات، أو لأداء مهام ضرورية اخرى فى ال event الخاص بال exception الذى حدث بواسطة عبارات فى ال try block السابق. تبدأ ال exception raised بعبارات فى ال try block السابق. تبدأ ال Finally blocks بالكلمة الاساسية finally ويجب ان يتبعوا ال try block مباشرة. لا تعالج ال exceptions Finally blocks فقط ال except blocks يمكن ان تفعل هذا.

## الباب التاسع عشر : التعامل مع ال exceptions

● **Protected-statement block**، هي عبارة واحدة أو أكثر لها handlers خاص لاي exceptions يمكن ان ترفعها تلك العبارات . ان ال protected-statement block يتكون من زوج من ال try block وال except .

● **Protected-resource block**، هي عبارة واحدة أو أكثر لها deallocation handlers معينة لاي exceptions يمكن ان تحدثها استخدام ال resources . و امثلة resources تشمل تخصيص أماكن للذاكرة، ال Windows resources ، والملفات . يتكون ال protected-resource block من زوج من ال try block وال finally .

● **Robust application**، هو برنامج يستخدم blocks عبارات محمية و resource محمية لمعالجة كل ظروف الخطأ الممكنة التي قد تحدث، وحتى يزيل تخصيص ال resource بسلام .

**ملحوظة:** تذكر هذه القاعدة : ان robust application يأخذ وقتاً فقط وهو لا يترك آثاراً . والهدف من استخدام exceptions هو كتابة code قوى تتولى بعناية كل ظروف الخطأ الممكنة .

Note

### ال Protected-statement blocks.

أداتك الأساسية لمعالجة ال exceptions . توضح القائمة (١٩-١) التخطيط لإنشاء protected-statement block باستخدام الكلمات الأساسية try و except . لاحظ ان ال block ينتهى بالكلمة الأساسية end وفصلة منقوطة . وتشير علامات التعليق تشير الى نوع العبارات التي يمكنك ادخالها فى المواضع المختلفة .

(القائمة للمعلومات فقط وليست على القرص المدمج) .

القائمة (١٩-١)، تخطيط لإنشاء protected-statement block

```
{ Unprotected statements }
try
{ Protected statements that may raise an exception }
except
{ Statements to handle any raised exceptions }
end;
{ More unprotected statements }
```

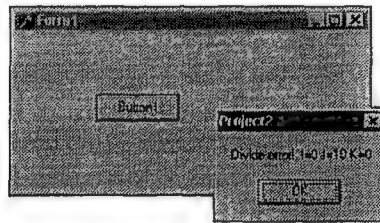
جرب مثلاً حقيقياً لتعرف كيفية استخدام التخطيط الموجود فى القائمة (١٩-١) وقم بإنشاء protected-statement block . اتبع الخطوات التالية:

١- ابدأ تطبيقاً جديداً.

٢- أضف Button object على ال form .

٣- اضغط مرتين Button1 ، واستخدم القائمة (١٩-٢) كإرشاد فى كتابة برنامج OnClick مع protected-statement block . ان نص هذه القائمة موجود فى ملف Except1.pas على القرص المدمج فى دليل ال Source\ExceptMisc .

٤- قم بـ Compile وتشغيل البرنامج بضغطة F9 . اضغط زر البرنامج لتجبر ال exception على الحدوث . يوضح شكل (١٩-١) ال رسالة المعروضة بواسطة عبارة ال ShowMessage exception handler التى تراها عندما تضغط الزر.




شكل (١٩-١): ال OnClick لـ Except1.pas يعرض هذه الرسالة عندما يتعامل exception مع القسمة على صفر

القائمة (١٩-٢): ExceptMisc\Except1.pas ان OnClick Button هذا يعرض كيفية إنشاء protected-statement block باستخدام ال try وال except

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K: Integer;
begin
  I := 0;
  J := 10;
  try
    K := J div I;
```

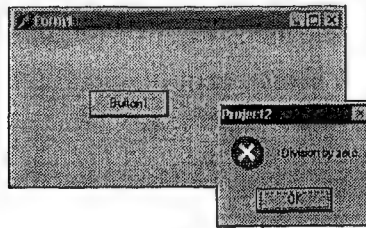
```
except
  ShowMessage(
    'Divide error! '+
    'I=' + IntToStr(I) +
    ' J=' + IntToStr(J) +
    ' K=' + IntToStr(K) );
end;
end;
```

**Tip**  **فكرة:** عند اختيار ال exceptions، من الأفضل ان تغلق message-spy utility WinSight32 التابعة لـ Delphi اذا كانت تعمل . اذا كنت تستخدم Delphi 4، اختر خيارات ال Tools|Debugger، اختر صفحات ال exceptions اللغة، وألغ اختيار الخيار "Stop on Delphi Exceptions" . اذا كنت تستخدم Delphi 3، اختر ال Preferences page tab، وابطل الخيار Break on Exception . كذلك انظر فصل "اعتبارات البيئة" في هذا الباب لمعرفة المزيد عن استخدام هذه الخيارات لازالة أخطاء exception handling البرنامج . ولكن اثناء تعلم كيفية exception، من الأفضل ان تبطل وسائل تقرير ال exception الزائدة هذه . بهذه الطريقة، إنك ترى بدقة كيف يستجيب البرنامج لنظام المستخدم النهائي . (بالتبادل، يمكنك compile البرنامج وتشغيل ملف ال exe الخاص به باستخدام ال Windows Explorer).

ان ال procedure الموجودة في القائمة (١٩-٢) ينفذ معادلة رياضية لأكل البازلاء بالسكين . ان التعبير  $J \div I$  يحاول ان يقسم عشرة على صفر، مما يجعل ال run-time library ال Object Pascal تحدث exception . ولكن لان التعبير موجود في try block، ال procedure يعالج ال exception بعرض رسالة . ان عبارة ال ShowMessage في ال except block تنفذ فقط اذا كانت عبارة واحدة أو أكثر في ال try block السابق فاشلة . يمارس البرنامج تحكما تماماً في أى exception - اذا دخل البرنامج ال except block، تكون قيمة ال K غير معرفة بالطبع، هذا مجرد مثال بسيط، ولكن في ال code المتداخلة بعمق والمعقدة تكون مثل هذه المعلومات لقيمة لها .

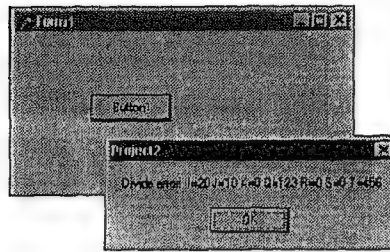
لعرض ما يحدث عندما تحدث ال unprotected-statement exception، احذف ال try، ال except وال end الأولى من القائمة (١٩-٢). اترك العبارات الأخرى كما هي، ثم اضغط F9 لـ compile وتشغيل البرنامج المعدل. اضغط الزر لتري dialog الذي يعرضه برنامج ال exception handler الافتراضى حسب نظام Delphi، والموضح فى شكل (١٩-٢). هذه التجربة توضح انه حتى اذا لم تستخدم ال protected-statement blocks، يتم معالجة كل ال exception فى النهاية. ان الاستجابة الافتراضية حسب النظام لتلك ال exception قد لا تكون كافية لتطبيق، ولكن على الاقل ان البرنامج ليس على اسوأ حال بلا قيمة. ان الاستجابة الافتراضية تكون دائماً افضل من لا شئ.

**على القرص المدمج:** عندما يحدث exception فى ال try block، يقفز التنفيذ على الفور الى العبارة الاولى فى ال try block. لا يتم تنفيذ العبارات الأخرى فى ال except block بعد تلك التى exception، وهذه الحقيقة قد يكون لها عواقب هامة. على سبيل المثال، فكر فى ال protected block فى القائمة (١٩-٣)، والذي يمكنك ادخاله فى ال OnClick Button كما فى الاختبار السابق. (يوجد ال procedure على القرص المدمج فى دليل Source\ExceptMisc فى ملف Except2.pas). اذا فشل أى من تعبيرات القسم لأى سبب، يقفز التنفيذ على الفور للـ ShowMessage، بعدها يستمر ال procedure بشكل طبيعى بعد نهاية ال protected block. على سبيل المثال، اذا فشلت القسم الثانية، وهذا ماسيحدث مع القيم المعينة المتغيرات ال procedure، لا تنفيذ العبارة الثالثة فى ال try block. يوضح شكل (١٩-٣) ال dialog الرسالة المعروضة بواسطة عبارة ال ShowMessage للـ procedure.



شكل (١٩-٢): برنامج ال exception handler الافتراضى لتطبيق Delphi يعرض displays لأى unhandled exceptions

## الباب التاسع عشر : التعامل مع الـ exceptions



شكل (١٩-٣): برنامجOnClick Except2.pas يعرض هذه الرسالة عندما يحدث exception

القائمة (١٩-٢): ExceptMisc\Except2.pas. اذا أحدثت أي واحدة من عبارات

القسم الثلاثة exceptions، يقفز التنفيذ مباشرة الى الـ ShowMessage

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
```

```
    I, J, K, Q, R, S, T: Integer;
```

```
begin
```

```
    I := 20; J := 10; Q := 123; T := 456;
```

```
    try
```

```
        K := (I div J) - 2;
```

```
        R := Q div K; // ???
```

```
        S := T div R;
```

```
    except
```

```
        ShowMessage(
```

```
            'Divide error! '+
```

```
            'I=' + IntToStr(I) +
```

```
            'J=' + IntToStr(J) +
```

```
            'K=' + IntToStr(K) +
```

```
            'Q=' + IntToStr(Q) +
```

```
            'R=' + IntToStr(R) +
```

```
            'S=' + IntToStr(S) +
```

```
            'T=' + IntToStr(T) );
```

```
    end;
```

```
end;
```

على القرص المدمج: تعرض القائمة (١٩-٣) واحدة من المميزات الرئيسية لمعالجة ال exception - لزاله الخطأ المتكرر بفحص العبارات الفورية المرتبطة السابقة أو التالية. من المفيد ان نقارن التشابه بين ال error-handling و ال non-exception، والذي يوجد له objects لا تحصى. توضح القائمة (١٩-٤) حالة فوضوية نموذجية. لا تكتب ابدأ code مثل هذا. يوجد ال procedure على القرص المدمج فى ملف ال Except3.pas فى دليل Source\ExceptMisc. يمكنك اختباره باستخدام procedure مثل ال Button OnClick يعرض ال procedure displays الرسالة نفسه الموضح فى شكل (١٩-٣).



القائمة (١٩-٤): ExceptMisc\Except3.pas مع exception handling إنك لا تحتاج الى كتابة code فوضوية مثل هذه

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K, Q, R, S, T: Integer;
{ Sub-procedure for displaying error message }
procedure ReportError;
begin
  ShowMessage(
    'Divide error! '+
    ' I=' + IntToStr(I) +
    ' J=' + IntToStr(J) +
    ' K=' + IntToStr(K) +
    ' Q=' + IntToStr(Q) +
    ' R=' + IntToStr(R) +
    ' S=' + IntToStr(S) +
    ' T=' + IntToStr(T) );
end;
begin
  I := 20; J := 10; Q := 123; T := 456;
  if J = 0 then ReportError else
  begin
    K := (I div J) - 2;
```

## الباب التاسع عشر : التعامل مع ال exceptions

```
if K = 0 then ReportError else
begin
  R := Q div K;
  if R = 0 then ReportError else
  begin
    S := T div R;
  end;
end;
end;
end;
```

### اسئلة نموذجية:

- فيما يلي ثلاث نقاط يجبن عن بعض الاسئلة النموذجية والتي قد تكون لديك على كيف تؤثر ال protected-statement blocks على سير البرنامج :
- اذا كانت عبارة في ال exception block تعالج exception يستمر ال procedure أو ال function بشكل طبيعي بعد نهاية ال protected block .
- اذا لم تتولى أى عبارة exception handles يخرج ال procedure أو ال function الحالية فوراً، وير ال exception من خلال سلسلة الاستدعاء الى ان يجد ال handler المناسبة .
- ان ال Unhandled exceptions تصل فى النهاية الى ال handles exception الافتراضى للتطبيق، والذي يعرض dialog ورسالة . ان handles exception الافتراضى هو ال handles على أى exception لم توفر لها handles . لا يجب عليك ان تكتب exception handles الافتراضى حسب النظام- ان Delphi يلحقه تلقائياً بكل تطبيق عمل compile له . مؤخراً فى هذا الباب، سوف اشرح كيفية استبدال ال handles الافتراضى بـ code خاصة بك، والذي لا يكون ضرورياً، ولكن قد يكون نافعا فى المشروعات المتقدمة .

### : Protected-resource blocks

ان عرض رسائل الخطأ يعد جانباً واحداً فى ال exception handling ويجب ايضاً على ال robust application ان يستعيد الاستقرار عندما تحدث



الكارثة. على سبيل المثال، اذا حدث خطأ قرص، يجب ان يحرر التطبيق أى ذاكرة مخصصة والتي قد تشغل مساحة بلا داعى حتى يعيد المستخدم الاقلاع. يتخلص البرنامج القوى من الاخطاء باغلاق الملفات المفتوحة deallocating Windows resources، وفعل كل ما هو ممكن لاستعادة النظام فى وسط الفوضى.

استخدم الكلمات الاساسية try و finally لإنشاء protected-resource blocks. توضح القائمة (١٩-٥) التخطيط الاساسى، والذي يشبه الى حد كبير protected-statement blocks. فى الواقع، ان الاختلاف الوحيد هو فى كلمة finally مكان كلمة except. ولكن، بالرغم من التشابه الظاهري، تختلف protected-resource blocks بشكل كبير فى العمل عن protected-statement blocks. (نص هذه القائمة لغرض العرض فقط وليس موجود على القرص المدمج).

القائمة (١٩-٥)، تخطيط لإنشاء protected-resource blocks

```
{ Allocate memory or other resource }
try
    { Statements that may raise an exception }
finally
    { Free the resource-guaranteed to execute! }
end;
{ Continue if no exceptions occurred in the try block }
```

ان العبارات الموجودة فى finally block تنفذ دائماً بغض النظر عما اذا كانت ايه عبارات فى try block تحدث exception. من الناحية النموذجية، أن العبارات الموجودة فى finally block تحرر الذاكرة ويغلق الملفات، تؤدى عمليات اخرى يجب القيام بها والتي تستعيد الاستقرار الى النظام عندما يقع exception.

**تحذير:** ان العبارات الواقعة خارج ال try block والتي تستطيع ان تحدث exception تجعل ال procedure أو ال function تخرج على الفور، متخطية ال finally block. لضمان تنفيذ ال finally block، ضع فى ال try block كل العبارات التي قد يحدث exception وتترك ال resource المخصص.



## الباب التاسع عشر : التعامل مع ال exceptions

ان التخطيط العام فى ال protected-resource blocks هى اين توضع العبارة التى تخصص resource . فبالرغم من ان هذا قد يحدث resource (على سبيل المثال ، اذا فشل تخصيص الذاكرة نتيجة نقص ال RAM) ، فإن عبارة التخصيص لا ينتمى الى ال try block . وهذه القاعدة غاية فى الاهمية . وإحدى الطرق لتذكر القاعدة هى ان تحفظ دائماً ان الغرض من ال finally block هو إزالة تخصيص ال resource ، وانت لا تريد ان تزيل تخصيص شئ لم يكن مخصصاً فى المقام الاول . لذلك ، يجب ان تضع عبارة التخصيص قبل ال try block . وداخل ذلك ال resource ، أدخل أى عبارات قد تحدث exception ، مما يجعل ال procedure أو ال function ان تخرج تاركة ال resource المخصص متأرجحاً فى مكان ما فى الذاكرة حتى يعيد المستخدم الاقلاع .

ويوضح مثلاً بسيطاً هذه التقنية الهامة ، ويظهر كيفية استخدام ال protected-resource blocks للحماية دون وجود resources متارجحة ، والتى قد تقضى على RAM فى نظامك . ان اختفاء الايقونات ، والنص الذى يتحول بطريقة غريبة الى ال System font ، ولوحة المفاتيح الغير مستجيبة ، والفوضى العامة هى اعراض تسرب ال resources ، يحدث عادة بواسطة تأرجح الذاكرة وتخصيصات ال resources الاخرى .

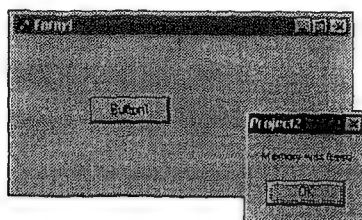
جرب هذه الخطوات :

١ - ١ - ابدأ تطبيقاً جديداً .

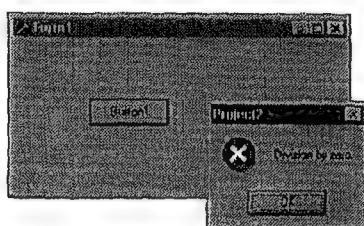
٢ - أضف Button على ال form .

٣ - اضغط ال Button مرتين لإنشاء OnClick ، وإملأ ال procedure باستخدام القائمة (١٩-٦) كمرشد لك . (هذا النص موجود على القرص المدمج فى ملف ال Except1.pas فى دليل ال Source\ExceptMisc) .

٤ - اضغط ال F9 ل Compile وتشغيل البرنامج ثم اضغط زر لاختبار ما يحدث فإذا حدث exception فى ال procedure يخصص قالباً من الذاكرة . شكل (١٩-٤) وشكل (١٩-٥) يظهر ال dialog الرسالة التى تراها بالترتيب عند تشغيل ال procedure الاختبارى .



شكل (٤-١٩): بالرغم من ان ال exception يحدث في ال procedure الاختباري Except1.pas ، الا ان dialog الرسالة هذا وذلك الموضح في شكل (٥-١٩) يثبت ان الذاكرة المخصصة قد تم تحريرها بشكل سليم



شكل (٥-١٩): يعرض ال Except1.pas هذا ال dialog بعد ان تفلق ذلك الموضح في شكل (٤-١٩)

القائمة (٦-١٩): ExceptMisc\Except4.pas . OnClick Button . هذا يوضح

كيفية إنشاء protected-resource block باستخدام ال try وال finally

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
```

```
  I, J, K: Integer;
```

```
  P: Pointer;
```

```
begin
```

```
  I := 0;
```

```
  J := 10;
```

```
  GetMem(P, 4098); // Allocate memory resource
```

```
try
```

```
  K := J div I; // Raises an exception!
```

```
  ShowMessage('Results: ' +
```

```
    ' I=' + IntToStr(I) +
```

```
    ' J=' + IntToStr(J) +
```

```
    ' K=' + IntToStr(K) );
```

## الباب التاسع عشر : التعامل مع الـ exceptions

```
finally  
    FreeMem(P, 4098); // Guaranteed to execute  
    ShowMessage('Memory was freed');  
end;  
end;
```

تشبه القائمة (١٩-٦) الـ protected-statement block، ولكن تضيف عبارة تخصيص ذاكرة تستدعي الـ GetMem للاحتفاظ بـ ٤٠٩٨ بايت من الـ RAM لاحظ ان هذه العبارة (وكذلك تعيينات متغير العدد الصحيح الغير ضارة) قد تم وضعها خارج الـ try block. وداخل الـ try block يوجد تعبير القسمة الخاطئة. اذا لم تكن هذه القسمة داخل الـ try block، واذا كان تنفيذ تلك العبارة تحدث exception، فقد ينتهي البرنامج على الفور، تاركا block الذاكرة المخصصة ضائعاً في الفضاء.

ان الـ finally block يحمي من هذا الخطأ باستدعاء الـ FreeMem بغض النظر عما اذا كانت القسمة تحدث exception ام لا. (قم بتعيين اثنين للـ I واعد تشغيل الاختبار لتثبت ذلك). ان عبارة الـ ShowMessage الاخيرة هي لمجرد الايضاح- عملياً، تقوم الـ finally blocks غالباً بخدماتها في هدوء.

**ملحوظة:** تذكر : ان العبارات في الـ finally block مضمون تنفيذها حتى اذا تم رفع أي exception في الـ try block السابق.

ان الـ finally blocks لا تعالج الـ exception فقط ولكن الـ except block هي تفعل هذا. الـ finally blocks يضمن تنفيذ عبارات بغض النظر عما اذا كانت عبارات اخرى موجودة في الـ try blocks السابق تحدث exception ام لا. في هذه الحالة، لا يتم الـ exception handler، وبذلك، عندما تقوم بتشغيل الـ code في القائمة (١٩-٦)، يعرض برنامج الـ exception handler الافتراضى لـ Delphi dialog رسالة خطأ. ولكن، كما ستوضح الفصول التالية، يمكن للـ procedure ان يكون لديه protected-statement و protected-resource blocks.

## ١١ try-except وال try-finally blocks المتداخلة:

إذا كنت مازالت فى بداياتك مع ال exception handler ، فقد تتساءل كيف يمكنك ال exception handler وحماية تخصيصات ال procedure فى نفس ال procedure . والحلية هى ان تذكر ان هذين يعتبران عمليتان منفصلتان تماماً ، وانك لا يمكن ان تدمج ال try-except وال try-finally blocks . فلا يوجد object مثل ال try-except-finally block فى ال Object Pascal الخاص بـ Delphi .

ولكن ، يمكنك ان تدخل ال try-except block فى ال try-finally blocks لتعالج ال exceptions وتمنع تأرجح ال resources . توضح القائمة (١٩-٧) التخطيط الاساسى لانجاز هذا المستوى الاضافى من الحماية . (هذا . ال code للعرض فقط وغير موجودة على القرص المدمج) .

القائمة (١٩-٧): تخطيط لإنشاء ال protected-statement و protected-resource

### blocks متداخلة

```
{ Allocate resource }
try
  try
    { Statements that might raise exceptions }
  except
    { Statements that handle exceptions }
  end;
finally
  { Free the resource }
end;
```

إنك تحتاج الى استخدام ال try-except block وال try-finally block المتداخلة كما هو موضح فى القائمة (١٩-٧) فقط عندما تعالج ال except block نوع معين من ال exception ، وفى هذه الحالة ، فان الانواع الاخرى من ال exception تجعل ال procedure ينتهى مبكراً . فى هذا المثال والامثلة السابقة ، يعالج ال except block كل ال exception التى تحدث فى ال try block السابق . ولكن كما

## الباب التاسع عشر : التعامل مع ال exceptions

سأوضح مؤخراً في هذا الباب ، إنك غالباً ما تقبض على بعض ال exception وتترك الأخرى تمر إلى أعلى في سلسلة الاستدعاء ، وربما تصل إلى برامج ال exception المعالجة الافتراضية لـ Delphi في غياب ال try-finally block ، فإن تلك ال exception الأخرى ، لأنها لم يتم علاجها في ال procedure ، قد تجعل تخصيصات الذاكرة لا يتم تحريرها .

**على القرص المدمج: مؤخراً في هذا الباب ،** سوف أشرح كيفية معالجة أنواع معينة من ال exception باستخدام عبارات ال on-do . أما الآن ، فإن القائمة (٨-١٩) توضح مثال قابل للتنفيذ للتخطيط الوارد في القائمة (٧-١٩) . توضح القائمة (٩-١٩) نفس ال code مع ال try-except block المتداخل وقد تم إبعاده إلى function قابلة للاستدعاء ، والتي تعتبر أقل فاعلية ، ولكن تلغى التداخل الذي يجده بعض المطورين أمراً مجيراً . جرب كلا ال two procedures كما فعلت فيما سبق بإضافة Buttons على form وإنشاء OnClick . اضغط F9 لل compile والتشغيل ، ثم اضغط الزر . تثبت رسائل البرنامج أن الذاكرة المخصصة قد تم تحريرها بالرغم من أن هناك exception يحدث . توجد القائمتان في ملفي Except5.pas و Except6.pas وعلى القرص المدمج في دليل Source\ExceptMisc .

أن Button OnClick في القائمة (٨-١٩) يظهر كيفية تداخل ال try-except block في ال try-finally block لمعالجة ال exceptions ومنع ال resources من التراجع .

القائمة (٨-١٩) ، ExceptMisc\Except5.pas

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K: Integer;
  P: Pointer;
begin
  I := 0;
  J := 10;
  GetMem(P, 4098);
  try
```

```

try
    K := J div I;
    ShowMessage('Results: K=' + IntToStr(K));
except
    ShowMessage('Divide error! ' +
        ' I=' + IntToStr(I) +
        ' J=' + IntToStr(J) );
end;
finally
    FreeMem(P, 4098);
    ShowMessage('Memory was freed');
end;
end;

```

ان OnClick و function فى القائمة (٩-١٩) يعتبران من الناحية العملية متطابقان لل code الموجودة فى القائمة (٨-١٩)، ولكنها تصنع ال try-except المتداخل داخل function قابلة للاستدعاء، وهى ال GetInt. والنتيجة هى OnClick procedure اوضح على حساب استدعاء function اضافى.

**القائمة (٩-١٩) ExceptMisc\Except6.pas**

```

function GetInt: Integer;
var
    I, J, K: Integer;
begin
    I := 0;
    J := 10;
    try
        K := J div I; // Raises an exception
        Result := K; // Assign function result (doesn't
        execute)
        ShowMessage('Results: K=' + IntToStr(K));
    except
        Result := 0; // Assign function result on error
    end;
end;

```

## الباب التاسع عشر : التعامل مع ال exceptions

```
ShowMessage('Divide error! ' +  
  ' I=' + IntToStr(I) +  
  ' J=' + IntToStr(J) );  
end;  
  
end;  
  
procedure TForm1.Button1Click(Sender: TObject);  
var  
  K: Integer;  
  P: Pointer;  
begin  
  GetMem(P, 4098); // Allocate memory resource  
  try  
    K := GetInt; // Might cause an exception  
  finally  
    FreeMem(P, 4098); // Guaranteed to execute  
    ShowMessage('Memory was freed');  
  end;  
end;
```

توضح القائمة (٩-١٩) كيف تقوم معالجة ال exception بفك ازدواج عبارات برنامج عادية من منطق معالجة الخطأ الخاص بها. على سبيل المثال، احذف try، و except، وكل العبارات في قالب ال except وال and الأولى من ال GetInt function. تبدو ال function مثل هذه:

```
begin  
  I := 0;  
  J := 10;  
  K := J div I; // Raises an exception and exits  
procedure!  
  Result := K;  
  ShowMessage('Results: K=' + IntToStr(K));  
end;
```



اضغط F9 ولإعادة تشغيل البرنامج، ثم اضغط الزر لتخبر ال exception على الحدوث. ان ال Button OnClick مازال يحرق الذاكرة المخصصة بالرغم من ان ال function لم تعد تعالج ال exception الذى حدث بنسب القسمة الخاطئة. اذا كان OnClick لم يستخدم ال try-finally، فإن ال GetInt's exception قد يجعل ال OnClick procedures ينتهى، تاركاً الذاكرة المخصصة تتأرجح.

## معالجة ال Exception:

قد مت الفصول السابقة اغلب تقنيات معالجة ال exception اللازمة لكتابة اللازمة لكتابة تطبيقات Delphi قوية. فى هذا الجزء، سوف تتخطى الاساسيات لتعرف كيفية معالجة انواع خاصة من ال exception، وكيفية حدوثه وحدوث exception خاصة بك. ولكن اولاً، انك بحاجة الى التفكير فى بعض الخيارات التى تؤثر فى كيفية استجابة نظام التطوير الخاص بك لل exception.

عند تطوير تطبيقات - وخاصة debugging exception handlers - فمن الأفضل أن تجعل Delphi يوقف البرنامج عندما يحدث exception. لتشغيل هذا الميزة، اختر امر ال Tools| Environment Options... اذا كنت تستخدم Delphi 3، اختر ال Preferences page tab، وقم بتشغيل ال Break on Exception check box. اذا كنت تستخدم Delphi 4، اختر خيارات ال Tools|Debugger، اختر صفحة exception اللغة، وابطل اختيار ال Stop on Delphi Exceptions check box.

**ملحوظة:** بعض الاوامر المشار إليها هنا تختلف اعتماداً على نسختك من Delphi. اذا كنت تستخدم Delphi Version 1، اختر ال Environment|Options، وللـ Version 2، اختر

Note

Tools|Options، ثم اتبع التعليمات السابقة للـ Delphi Version 3.

حاول تشغيل برنامج اختبار exception (واحداً يحتوى على ال event handler للقائمة (١٩-٢٠)، مثلاً). مع اختيار وابطال هذه الخيارات. عند تشغيلها، يعرض الخيار تقريراً كاملاً عن أى exceptions، ويتوقف البرنامج عند العبارة التى سببت المشكلة. يوضح شكل (١٩-٦) ال dialog box الناتج الذى يعرضه Delphi. والسطر الذى سبب المشكلة يتم ابرازه فى نافذة محرر ال Delphi

## الباب التاسع عشر : التعامل مع ال exceptions

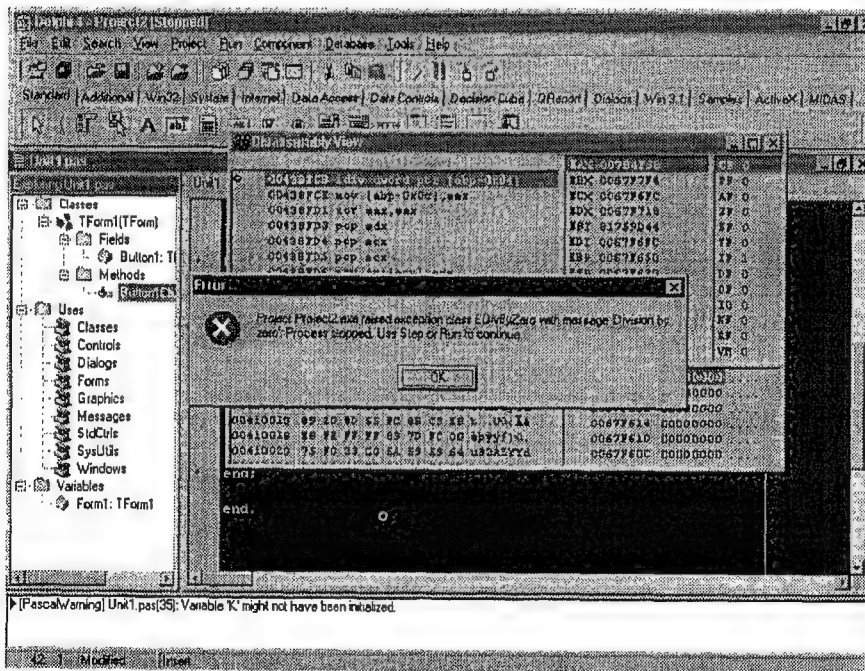
code . من المهم ان تفهم ان Delphi نفسه ، وليس برنامج معالجة ال exception الافتراضى للتطبيق ، يعرض نافذة ال dialog هذه .

يجب ان تقوم بتشغيل برنامجك داخل Delphi (بمعنى اوضح ، فى نمط ال dialog) لتراه . ان المستخدمين النهائيين لتطبيقك لا يرون ابدأ هذا ال dialog .

اقرأ معلومات ال dialog ، واضغط زر ال OK . اختبر العبارة التى تم ابرازها والتى أحدثت exception . مازال البرنامج يعمل ، ولكنه يتوقف عند العبارة المتهمة . اضغط F9 لاستمرار التشغيل ، أو يمكنك ان تضغط F8 أو F7 لتنتقل خطوة واحدة بالبرنامج بدءاً من هذا الموضع .

**فكرة:** بالرغم من انك تنظر الى code برنامجك فى Delphi ، فمازال البرنامج يعمل فى نمط ال dialog . يجب ان تقوم بتشغيل واخراج البرنامج قبل ان تستمر فى تطوير التطبيق .

Tip



شكل (١٩-٦) : عندما يتم معالجة ال exception بواسطة ال debugger ويحدث exception يتوقف البرنامج ويعرض Delphi displays يشبه هذا

للاستمرار، اضغط F9 الآن. لقد انتقلت مرة أخرى إلى نافذة البرنامج، واعتماداً على المثال الذى تقوم بتشغيله، فإنك ترى واحدة أو أكثر من يهشمخل الرسائل التى يتم عرضها بواسطة البرنامج نفسه. فى المرة الأولى التى تجرب فيها هذه الخطوات، فإن ال dialogs قد تبدو محيرة، ولكن بعد ان تفهم المستويات المختلفة لمعالجة ال exception التى تحدث، فإن المعلومات تصبح قيمة للغاية فى مطاردة الاخطاء- خاصة أى اخطاء فى برامجك المعالجة لل exception.

عندما يتم معالجة ال exception بواسطة ال debugger، يمكنك تشغيل التطبيق باستخدام ال Windows Explorer لاختبار كيف يستجيب البرنامج التام للاخطاء خارج Delphi. قم بتشغيل ال code داخل Delphi للحصول على معلومات حول ال exception بالإضافة الى أى معالجة ال exception التى يوفرها تطبيقك.

**ملحوظة:** لاستكمال هذا الباب، قد تود ان يكون لديك exception تعالج من خلال debugger، ولكنك تحتاج ان تضغط F9 مرة واحدة لتستمر فى تشغيل البرنامج فى كل مرة يحدث فيها ال exception. اذا وجدت ان هذا متعب لك، اغلق معالجة ال exception بال debugger. اذا كنت تستخدم Delphi 3، اختر ال Preferences page tab، وقم بإبطال ال Break on Exception check box.

### ال exception instance:

كما ذكرنا، ان exception هو حدث أو ظرف يعترض السير الطبيعى للبرنامج. ولكن من الناحية التكوينية، ال exception يعتبر object من class، مشتقة من ال Exception class المعروفة فى ال SysUtils unit. وهذا ال object يسمى exception instance.

ان حدوث exception يؤدي الى انشاء exception instance. اذا كانت عبارة فى جزء ال except من ال protected-statement block تعالج ال exception، فإن البرنامج يدمر ال exception instance بصورة تلقائية. اذا لم تتولى أى عبارة معالجة ال exception، يقوم التطبيق بتمرير ال exception instance الى أعلى فى سلسلة الاستدعاء حتى تجد معالجة مناسبة، أو حتى تصل الى برنامج exception handler الافتراضى حسب النظام.

## الباب التاسع عشر : التعامل مع ال exceptions

ان الطريقة الوحيدة الآمنة لتدمير exception instance هي معالجة ال exception . فمعالجة ال exception تدمر ال exception instance بصورة تلقائية . فى الفصول التالية ، سوف تتعلم كيفية الاشارة الى exception instance . لا تقم ابدأ بكتابة code لتحرر أو تدمر هذه ال objects . ان محاولة تحرير exception instance يؤدي الى خطأ فادح بالتطبيق .

### معالجة exception معينة:

ان exception instance تعرف نوع ال exception . الذى حدث . يوفر ال Delphi عدة classes ، كلها مشتقة من ال Exception ، والتي تصف طبيعة مشكلة معينة . يمكنك استخدام هذه المعلومة لتحسن استجابة برنامجك لل exception ، وقد تكتب برامج معالجة exception يتم اطلاقها بواسطة انواع معينة من المشكلات . لتنفيذ هذه المهام ، استخدم عبارة ال on-do داخل ال exception block .

تنفذ عبارة ال on-do خدمتين : إنها تعرف نوع معين من ال exception ، ويمكن ان توفر reference الى ال exception instance وتوفر ال exception instance معلومات عن نوع المشكلة التى حدثت . يمكن لبرنامجك ان يستخدم هذه المعلومات فى حل المشكلة ولعرض ملاحظات مساعدة للمستخدمين .

### الإشارة الى حالة exception instance :

**على القرص المدمج:** ان التطبيق Except1 على القرص المدمج فى دليل Source\Except1 يوضح كيفية استخدام ال on-do للإشارة الى exception instance يوضح شكل (١٩-٧) نافذة البرنامج ال dialog المعروف عندما تضغط الزر . هذا يجبر ال exception على الحدوث بمحاولة تعيين قيم غير مسموح بها لل scrollbars controls التابعين للنافذة . توضح القائمة (١٩-١٠) ال Button .

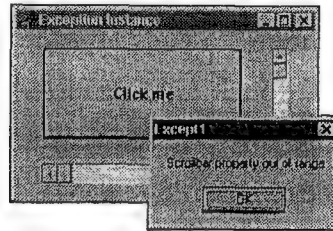
القائمة (١٩-١٠) Except1Main.pas ال onClick لل Button

```
procedure TMainForm.Button1Click(Sender: TObject);
begin
    try
```

```

ScrollBar1.SetParams(0, 500, 0);
ScrollBar2.SetParams(0, 500, 0);
except
on E: Exception do
    ShowMessage(E.Message);
end;
end;

```



شكل (١٩-٧): ان ضغط زر ال Except1 يجبر ال exception على الحدوث ويعرض dialog الموضح هنا

يستدعى البرنامج ال scrollbars SetParam ، ولكن لاجبار ال exception على الحدوث ، فإنه يعكس متعمداً قيم ال Max وال Min . يحدد ال ScrollBar ان ال Min يجب ان يكون اقل من أو مساو لل Max ، والاستدعاء الاول لل SetParams يحدث exception لهذا السبب .

فى ال except block ، تعرف عبارة on-do متغير E لل Exception class . (يمكنك استخدام أى اسم ، ولكن E اسم مناسب) . وهذه العبارة لا تنشئ حالة Exception جديدة ، إنها فقط تنشئ إشارة لل (E) لل exception instance التى تصف المشكلة . يستخدم البرنامج خاصية ال Message string لل E لعرض رسالة ذات معنى فى ال dialog box .

ولكن ، الايقاع بكل نوع exception بهذه الطريقة لا يكون دائماً افضل الحلول . والخطة الافضل من هذه هى الايقاع بـ exception معينة ، وترك الاخرى تمر بطريقة عادية . بهذه الطريقة ، تستجيب ال code لل exception التى تم اعدادها على ان تعالجها - parameter سىء ، مثلاً ، أو خطأ القسمة على صفر - ولكن تسمح برامج معالجة ال exception الافتراضية ل Delphi ان تعتنى بالمشكلات الاكثر خطراً مثل خطأ الخروج من الذاكرة .

## الامساك بانواع معينة من ال exception :

على القرص المدمج: يمكنك استخدام ال on-do للإيقاع بانواع معينة من ال exception ، مثلاً ، لمعالجة ال exception القسمية على صفر ، ولكن اسمح لبرامج معالجة أخرى أعلى في سلسلة الاستدعاء ان تعالج انواع ال exception الأخرى ، كما هو موضح في القائمة (١٩-١١) . هذه هي نفس ال code الموجودة في القائمة (١٩-٢٠) ولكن هذه المرة ، تعالج بشكل خاص فقط ال exception EDivByZero . على القرص المدمج ، يوجد نص ال procedure . في ملف Except7.pas في دليل ال Source\ExceptMisc .



القائمة (١٩-١١) ال OnClick Button يعرض كيفية انشاء ال protected-statement

block باستخدام ال try وال except لنوع معين من ال exception

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, J, K: Integer;
begin
  I := 0;
  J := 10;
  try
    K := J div I;
  except
    on E: EDivByZero do
      begin
        ShowMessage(E.Message +
          ' I=' + IntToStr(I) +
          ' J=' + IntToStr(J) +
          ' K=' + IntToStr(K) );
      end;
  end;
end;
```

لاحظ ان عبارة ال ShowMessage تعرض اولاً E.Message ، والتي تم تعريفها في كل ال exception objects على انها خاصية String . لا يجب عليك ان تعرض هذا ال string ، ولكنه متضمن في كل exception objects ، وفي اغلب الحالات ، يجب ان يظهر dialog الخطأ في مكان ما .

### اختبار exception classes :

فيما يلي قائمة exception classes المعرفة في ال Delphi units المتنوعة . هذه القائمة ليست شاملة ، ولكن تتضمن أغلب ال exception التي غالباً ماستستجيب لها أو ترفعها في تطبيقاتك . ان كل ال classes في الجدول مشتقة ، وليس مباشرة بالضرورة ، من ال Exception class . استخدمها كما توضح القائمة (١٩-١١) . على سبيل المثال ، للامساك بخطأ EPrinter ، استخدم code مثل هذه :

```
try
  { Printer operations }
except
  on E: EPrinter do
    ShowMessage(E.Message); // Display printer error
    message
end;
```

### ● EAbort : exception يحدث بواسطة استدعاء ال Abort .

● **EAbstractError** : يتم حدوثه اذا حاولت عبارة ان تستدعي abstract method . يعرض Delphi تحذيراً اذا قام ال code بانشاء instance من class ذات واحد أو أكثر من abstract methods . والمرة الوحيدة التي ستلقى فيها ال exception EAbstractError هي اذا تجاهلت هذا التحذير .

● **EAccessViolation** : يشير الى واحد من ثلاث أخطاء وصول غير مسموح الى الذاكرة : باستخدام ال nil pointer ، أو بالكتابة الى صفحة code memory ، أو بالوصول الى صفحة ذاكرة غير مخصصة . لا يحدث هذا ال exception ظاهرياً .

## الباب التاسع عشر : التعامل مع ال exceptions

● **EArrayError**: يشير الى واحد من ثلاث عمليات array غير مسموح بها: فهرس خارج النظام، محاولة اضافة عنصر الى array ذا حجم ثابت، ادخال عنصر في sorted array. انظر ايضاً EBitsError.

● **EAssertionFailed**: يحدث بواسطة ال Assert عند تمرير تعبير False Boolean.

● **EBitsError**: يشير الى واحدة من اثنين من عمليات Boolean arrays الغير مسموح بها (باستخدام ال TBits class): قيمة فهرس سالبه أو خارج النطاق.

● **EClassNotFound**: هذا ال exception يتم حدوثه عندما يحاول البرنامج قراءة object من class غير مرتبطة بالتطبيق (مثلاً، تم حذف تعريفها من تركيبه source code لل form، أو حاول البرنامج قراءة object غير معروف من مجموعة).

● **EComponentError**: يحدث عند أى خطأ اثناء تسجيل ال component يحدث ايضاً اذا حاول البرنامج ان يحصل على واجهة تطبيق ال COM لل component اذا لم يكن هذا ال component يؤيد ال COM. انظر الباب العشرين لمزيد من المعلومات عن انشاء وتشغيل ال component.

● **EControlC**: يخص التطبيقات فقط. هذا ال exception يتم حدوثه اذا قام مستخدم البرنامج بضغط Ctrl+C. هذا ال exception لا يحدث ابداً في تطبيق Windows.

● **EConvertError**: يحدث لاختفاء التحويل مثل محاولة تحويل string ذا رمز غير رقمي الى عدد صحيح باستدعاء ال StrToInt.

● **EDatabaseError**: تحدث exception نتيجة لخطأ قاعدة البيانات العام بواسطة ال Data Access وال Data Controls. لمعرفة المزيد عن بعض exception قاعدة البيانات، ابحث في online help التابعة ل Delphi عن ال EDB exception classes، مثلاً، EDBClient و EDBEditError.



• **DateTimeError**: يحدث بواسطة اوقات وتواريخ غير صحيحة يتم ادخالها فى الـ `TDateTimePicker object`.

• **EDivByZero**: عدد صحيح يحاول القسمة على صفر.

• **EFCREATEERROR**: يشير الى خطأ فى انشاء ملف.

• **EFileError**: يحدث بسبب المشاكل التى تحدث اثناء عمليات الـ `file-stream`.

• **EOutError**: خطأ فى إدخال/إخراج ملف. ان حقل الـ `Integer ErrorCode` exception instance يحمل code خطأ I/O. وهذا الـ exception يتم حدده فقط فى الـ `code compiled` مع مفتاح الـ `{ $I+ }` فى حالة التشغيل (هذا هو التحديد الافتراضى). والـ `codes` الخاطئة هى: ٢- ملف غير موجود، ٣- لاسم ملف غير صحيح، ٤- لملفات عديدة مفتوحة، ٥- `access denied`، ١٠٠- نهاية ملف، ١٠١- قرص ممتلى، و ١٠٦- أذخال بيانات غير صحيح. والـ `code` الخاطئة الاخرى ممكنة- انظر كتيب الـ `MS-DOS Technical Reference` لـ `Microsoft` ومصادر وثائق الـ `Win32` المتنوعة.

• **EIntError**: class اساسية لـ `exceptions` لحساب العدد الصحيح. لا يتم انشاءها ابداً كـ `exception object`، ولكن يمكن استخدامها للايقاع بكل خطأ عمليات الاعداد الصحيحة. لانشاء `exception objects` لاعداد صحيحة معينة، استخدم بدلاً من هذه الـ `exception classes` المشتقة `EIntOverflow`، `ERangeError`، و `EIntOverflow`.

• **EIntfCastError**: يحدث عندما تحاول عبارة ان تصنع class غير مناسبة مع عامل الـ `as`.

• **EIntOverflow**: قيمة العدد الصحيح أكبر من مداه المحدد.

• **EInvalidArgument**: قيمة خارج النطاق فى function محاسبة مالية فى وحدة الـ `Math`.

• **EInvalidCast**: تغيير `type-cast` غير صحيح.

## الباب التاسع عشر : التعامل مع الـ exceptions

● **InvalidGraphic**؛ يحدث عندما يحاول البرنامج ان يفتح ملف جرافيك غير معروف (مثلاً، يفتح عن طريق الخطأ ملف نص كـ bitmap).

● **InvalidGraphicOperation**؛ يشير الى عملية جرافيك غير مسموح بها مثل محاولة تغيير حجم ايقونة.

● **InvalidGridOperation**؛ يحدث لعمليات الـ grid الغير مسموح بها مثل محاولة الوصول الى خلية غير موجودة.

● **InvalidOp**؛ عملية برنامج معالجة حسابية لـ floating-point غير صحيحة. هذا الـ exception يحدث للاخطاء الرياضية الغير معرفة عندما، مثلاً، تكتشف الـ CPU أمر غير معرف، تحاول اداء عملية غير صحيحة.

● **InvalidOperation**؛ يشير الى عملية غير صحيحة في component، وبخاصة واحدة تتطلب الـ window handle تكون خاصية الـ Parent له غير معينة. يمكن ان يحدث ايضاً لعمليات السحب والاسقاط الغير مسموح بها.

● **InvalidPointer**؛ pointer غير صحيح، مثلاً، محاولة استخدام nil pointer أو ترك block ذاكرة مخصصة أكثر من مرة.

● **EMathError**؛ class اساسية لـ exceptions حساب الـ floating-point. لا تستخدمه مباشرة كـ exception object استخدم بدلاً منه exception classes المشتقة من EInvalidOp، EInvalidArgument، EZeroDivide، EUnderflow، EOverflow.

● **EMCIDeviceError**؛ يحدث لمشكلات مع الـ Media Control Interface (MCI).

● **EMenuError**؛ يشير الى عملية غير مسموح بها في الـ menu.

● **ENoResultSet**؛ يحدث بواسطة objects قاعدة بيانات الـ Query المستخدمة بدون اشتراط عبارة SELECT.

• **EOleCtrlError**: يحدث عندما تحاول Ole فاشلة ان ترتبط بـ ActiveX control.

• **EOleException**: يحدث عند أى فشل مع استدعاء الـ IDispatch method.

• **EOleSysError**: يحدث اذا فشل method استدعاء الـ IDispatch.

\* **EOutOfMemory**: عدم القدرة على تحديد أماكن الذاكرة المطلوبة.

• **EOutOfResources**: يحدث عند حدوث محاولة فاشلة لإنشاء Windows handle.

• **EOverflow**: عدم تكون قيمة الـ floating-point أكثر من المدى المسموح به.

• **EPackageError**: يحدث لأى اخطاء تشمل الـ packages.

• **EPrinter**: يشير الى خطأ يحدث اثناء الطباعة.

• **EPrivilege**: يحدث لاططاء processor privilege مثل محاولة تنفيذ أمر غير مسموح فى مستوى الـ privilege الحالى.

• **EPropertyError**: يحدث اذا لم يكن ممكناً تحديد قيمة خاصية، على سبيل المثال، اذا كانت خارج النطاق أو نوع بيانات غير مسموح به.

• **EPropReadOnly**: يشير الى محاولة غير مسموح بها لكتابة خاصية قراءة فقط باستخدام الـ OLE.

• **EPropWriteOnly**: يشير الى محاولة غير مسموح بها لقراءة من خاصية كتابة فقط باستخدام الـ OLE.

• **ERangeError**: قيمة خارج النطاق لعملية فهرسة array أو short-string، وكذلك للتعينات لمتغيرات النطاق الفرعى. هذا الـ exception لا يتم حدوثه للـ long strings. انه يتطلب ان يكون البرنامج قد حدث له compiled مع مفتاح الـ { \$R+ }، وهذا ما يحدث فقط اثناء ازالة الاخطاء. ان

## الباب التاسع عشر : التعامل مع ال exceptions

التحديد الافتراضى { \$R+ } يغلق فحص المدى ، ويمنع هذا النوع من ال exception من ان يحدث .

• **EReadError** : يحدث عند محاولة خاطئة لقراءة بيانات خط ، وايضاً اذا كان ال Delphi غير قادر على قراءة قيمة خاصة اثناء انشاء form object .

• **ERegistryException** : يشير الى فشل لعملية تم اداءها على ال Windows registry مثل محاولة تعديل مساحة من السجل مع امتيازات مستخدم غير كافية .

• **EResNotFound** : يحدث اذا كان resource معين مثل الايقونة غير موجود . والعنصر الاساسى فى هذه المشكلة هو امر { \$R \*.DFM } الذى تم حذفه خطأ فى قطاع ال implementation فى source code module لل form unit .

• **ESocketError** : يشير الى فشل مع ال Windows socket object .

• **EStackOverflow** : يحدث عندما تنمو ال thread stack الحالية فى "final guard page" ، مما يشير الى ان البرنامج أوشك على على ان يفتقر الى الذاكرة والاسباب لهذه المشكلة هى المتغيرات المحلية الكبيرة جداً فى ال procedures و functions وكذلك فى النظم الفرعية المتكررة المتداخلة . (انقل المتغيرات الكبيرة خارج ال procedures وال functions .

• **EStreamError** : يشير الى خطأ مع عملية streaming مثل تحديد index غير صحيح ل list-box .

• **EThread** : يشير الى مشكلة تزامن ال thread .

• **ETreeViewError** : يحدث عندما يتم استخدام index غير مسموح به مع TreeView .

• **EUnderflow** : عندما تكون قيمة العمليات لل floating-point أقل من المدى المسموح به

• **EVariantError** : يشير الى خطأ مع نوع بيانات المتغير مثل type-cast غير صحيح ، أو index خارج المدى .

● **EWin32Error**، يتم حدوث هذا exception اذا تم اكتشاف خطأ Windows. يعرض برنامج معالجة ال exception الافتراضى dialog يذكر قيمة ال code الخاطئة ورساله تحديد الخطأ. يمكنك استخدام ال SysUtils Win32Check function مع هذا ال exception لتبحث عن الرساله من نظام التشغيل.

● **EWriteError**، يحدث لاططاء كتابة ال file-stream.

● **EZeroDivide**، Floating-point تحاول القسمة على صفر.

### البحث عن خصائص Exception Class:

ابحث عن انواع معينة من exceptions فى ال online help الخاصة بـ Delphi، ثم ابحث عن التعريف الخاصة لها. هذا يخبرك عن نوع- الخصائص التى تعرفها ال exception class. على سبيل المثال، تقوم ال SysUtils بتعريف ال EInOutError كما يلى:

```
EInOutError = class(Exception)
```

```
public
```

```
ErrorCode: Integer;
```

```
end;
```

هذا التعريف يخبرك ان، بالإضافة الى خاصية ال Message التى توفرها كل ال exceptions، يوفر ال EInOutError ايضاً حقل ErrorCode. ان ملف ال code ال Sysutils.pas source موضوع فى دليل تركيب Delphi وهو Source\RTL\Sys. يتم تعريف exception classes معينة فى كثير من ملفات ال source code الاخرى لـ Delphi.

ويمكنك ايضاً استخدام هذه ال class لأغراض التعريف الخالصة دون انشاء إشارة لل exception instance. على سبيل المثال، قم باستبدال عبارة ال on-do فى القائمة (١٩-١١) بما يلى:

```
on EDivByZero do  
    ShowMessage('Divide error');
```

يقارن السطر الاول exception instance class بـ EDivByZero، واذا كان ال object من هذه ال class يستدعى ال ShowMessage وتنتقل ال

## الباب التاسع عشر : التعامل مع الـ exceptions

exception من الانواع الاخرى الى أعلى من خلال استدعاء السلسلة . استخدم هذه التقنية اذا لم تكن تحتاج الى الاشارة الى الـ exception instance أو استخدم هذه الـ class :

```
on E: EDivByZero do
    ShowMessage(E.Message);
```

لعمل إشارة لـ E exception instance وعرض message string المحتوى فيها .

### معالجة الـ exceptions، المتعددة:

ان الـ except block قد يستجيب لأكثر من نوع من الـ exception . على سبيل المثال ، توضح جزئية الـ class التالية كيف معالجة ثلاثة انواع من الـ exception object ؛ تنتقل الانواع الاخرى لأعلى السلسلة حتى تجد معالم :

```
on EDivByZero do
    ...;
on EInOutError do
    ...;
on EOutOfMemory do
    ...;
```

يمكنك ايضاً تعريف exception instances للتوصل الى معلومات اخرى متنوعة متوفرة مع أنواع exception معينة . على سبيل المثال ، تمسك جزئية الـ code التالية بثلاثة انواع من الـ exception ، وتعرف exception instances لكل نوع :

```
try
{Statements that might raise one of the following
exceptions}
except
on E: EDivByZero do
    ShowMessage(E.Message + ' (I = 0)');
on E: EInOutError do
with E do
    ShowMessage(Message + ' #' + IntToStr
```

```

        (ErrorCode));
        on E: EOutOfMemory do
        begin
            ShowMessage(E.Message);
            // free reserved memory here
        end;
    end;
end;

```

ان كل exception handler فى الجزئية السابقة يستخدم المعلومات الموجودة فى الـ E بطريقة مختلفة. يسك المعالج الاول خطأ قسمة عدد صحيح على صفر، ويعرض خاصية الـ Message للـ E مع string حرفى ملحق بها يوفر قليلاً من المعلومات عما جرى من خطأ.

ويسك الـ exception handler الثانى باخطاء الـ I/O. إنه يستخدم عبارة with للتوصل لخصائص الـ E، ويعرض الـ string E.Message مع متغير الـ E.ErrorCode وقد تم تعريفه للـ exception class EInOutError. يتم تحويل هذه القيمة الى string مع استدعاء function IntToStr الخاصة بـ Delphi.

ويسك الـ exception handler الثالث باخطاء الـ out-of-memory. وهو يعرض الـ E.Message. ويشير التعليق الى مكان جيد لتحرير قالب الذاكرة المحجوز والذي قد يخصص البرنامج لتوفير بعض الذاكرة الاضافية، فقط لاغراض السماح للمستخدم بالخروج من البرنامج، أو لانتهاء العملية التى تسببت فى ظرف الـ out-of-memory.

**تحذير:** عندما تستخدم الـ on-do لمعالجة exception classes معينة، فإن وجود exception اخرى غير معالجة تجعل الـ procedure أو function الحالية تنتهى على الفور. فى مثل هذه الحالات، استخدم الـ try-finally block لتحرير الذاكرة المخصصة، لإغلاق الملفات، ولأداء عمليات اخرى واجبة الفعل.



### حدوث exception جديد:

يتم حدوث العديد من الـ exceptions بصورة تلقائية، على سبيل المثال خطأ القسمة على صفر، ولكن يمكنك ايضاً ان تحدث exception خاصة بك لتقرر اخطاء أو حالات غير عادية.

## الباب التاسع عشر : التعامل مع ال exceptions

استخدم الكلمة الاساسية raise عند حدوث exception جديد . بعد ال raise ، قم بانشاء exception instance من ال Exception class أو أي class مشتقة . بافتراض ان البرنامج له متغير ال Boolean وهو ErrorFlag الذي ، اذا كان True ، يشير الى مشكلة ، قد يحدث البرنامج exception بـ code مثل هذه :

```
if ErrorFlag then  
    raise Exception.Create('Error');
```

وبدلاً من ال Exception ، يمكنك استخدام اغلب ال classes المذكورة سالفاً . يمكنك ايضاً حدوث exceptions من classes خاصة بك ولكن سأوضح المزيد عن هذا فيما بعد . وتأتى نتائج هذه ال code مطابقة لل exceptions التى تحدثها ال components مثل أخطاء الآلة ، أو أى مصدر آخر . يمكنك كتابة برامج معالجة exception لل exception التى تحدثها ال code ، أو يمكنك ان تجعل برامج معالجة ال exception الاصلية تتولى امرهم .

**ملحوظة:** ان حدوث exception يبحث عن الفور عن اقرب exception فى ال procedure أو ال function التى تستدعى ال raise . اذا لم يكن لـ procedure أو function أية معالجة ، تنهى ال raise بفاعلية ال subroutine .

**تحذير:** كمثل على حدوث ال exceptions ، قم بتشغيل برنامج العرض Except2 على القرص المدمج فى دليل Source\Except2 ، ثم ادخل قيمة من صفر الى تسعة وتسعين فى نافذة ال Edit . يوضح شكل (١٩-٨) عرض البرنامج . اضغط زر ال I'm Done لانهاء البرنامج . قم بتشغيل البرنامج مرة اخرى وفى هذه المرة داخل قيمة غير مسموح بها مثل ١٠٠ . كما ترى ، هذا يؤدي الى حدوث exception يمنع البرنامج من الاغلاق . ان برامج معالجة ال exception الافتراضية لـ Delphi تعنى بتعريف المشكلة . توضح القائمة OnClick (١٢-١٩) للزر .

القائمة (١٢-١٩) : Except2\Main.pas لل Button

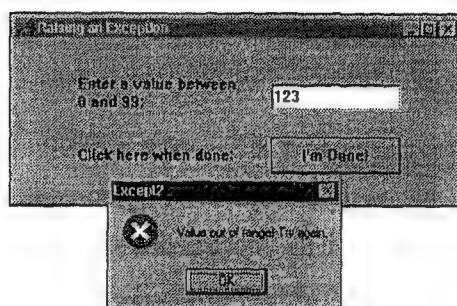
```
procedure TMainForm.Button1Click(Sender: TObject);  
var  
    N: Integer;
```



```

begin
  N := StrToInt(Edit1.Text);
  if (N < 0) or (N > 99) then
    raise ERangeError.Create('Value out of range! Try
      again')
  else begin
    ShowMessage('Success! Click Ok to end program');
    Close;
  end;
end;
end;

```



شكل (١٩-٨): إدخال قيمة خارجة عن المدى في نافذة الـ Edit للـ Except2 يؤدي إلى حدوث exception يمنع البرنامج من الانتهاء

في الـ Button OnClick، تحول الـ StrToInt خاصية الـ Text التابعة للـ Edit1 إلى قيمة عدد صحيح، وتعين هذه القيمة بـ N وترفع عبارة الـ if التالية exception إذا كانت الـ N ليست داخل المدى من صفر إلى ٩٩. بعد الكلمة الأساسية الـ raise، ينشئ البرنامج exception instance من الـ ERangeError class. في هذا البرنامج، يتلقى برنامج معالجة الـ exception الافتراضى الـ ERangeError ويعرض الرسالة في الـ dialog box.

ومن الجدير بالذكر أن الـ event handler قد يفشل بسبب نوع آخر من الـ exception: خطأ تحويل تم حدوثه بواسطة استدعاء الـ StrToInt. قم بتشغيل البرنامج مرة أخرى وادخل 3.14159 أو اسمك في نافذة الـ Edit، ثم اضغط الزر في هذه المرة ترى رسالة مختلفة مثل (3.14159 is not a valid Integer value) أو الرقم 3.14159 ليس قيمة صحيحة صالحة يتم عرض هذه الرسالة

## الباب التاسع عشر : التعامل مع ال exceptions

بواسطة برنامج معالجة ال exception الافتراضى ل Delphi . والاهم من هذا ان ال exception الذى يتم حدوثه بواسطة ال StrToInt يجعل ال event handler ينتهى على الفور ، مما يلغى باقى ال procedure code .

**على القرص المدمج:** وهناك نسخة افضل لل event handler هذا تستخدم ال try-except block لمعالجة هذا النوع من المواقف ، وتعطى المستخدم فرصة ليصحح الادخال الخاطئ ، بغض النظر عن السبب . قد تكون لاحظت انه عليك ان تضغط ال Tab أو تنقر الفأرة لتعود الى نافذة ال Edit بعد اغلاق message الخطأ . توضح القائمة (١٩-٣) الOnClick المأخوذة من مشروع ال Except3 على القرص المدمج فى دليل ال Source\Except3 .



يستخدم ال procedure الذى تم مراجعته ال procedure متداخلاً لعرض رسالة خطأ بالإضافة إلى الجملة Try again . ويستدعى أيضاً ال SetFocus ال method لل Edit1 لكى يجعل من السهل على المستخدمين ان يدخلوا قيمة اخرى بعد قراءة رسالة الخطأ . ان لب ال procedure يحدث exception اذا كانت ال N خارج المدى . وإنه يستخدم أيضاً ال try block بحيث ان الانواع الاخرى من ال exception كتلك التى يمكن حدوثها باستدعاء ال StrToInt يتم معالجتها هى الاخرى . ويوضح ال except block كيفية معالجة نوعين من المشاكل :

اخطاء التحويل التى يتم حدوثها بواسطة ال StrToInt ، واخطاء العدد الصحيح التى يتم حدوثها بواسطة ال procedure ذاته . فى كلتا الحالتين استدع ال Handler المتداخل لعرض الرسالة وحدد ال focus مرة اخرى الى ال Edit . وأى انواع اخرى من المشكلات يتم تمريرها الى برامج معالجة ال exception الافتراضية ل Delphi .

**القائمة (١٩-١٣): Except3\Main.pas لل Button1 الOnClick)**

```
procedure TMainForm.Button1Click(Sender: TObject);
var
  N: Integer; S: String;
  // Nested exception handler
  procedure Handler(Message: String);
  begin
    ShowMessage(Message + ' Try again.');
```

```

Edit1.SetFocus;
end;
begin
try
N := StrToInt(Edit1.Text);
if (N < 0) or (N > 99) then
raise ERangeError.Create('Value out of range!')
else begin
ShowMessage('Success! Click Ok to end
program');
Close;
end;
except
on E: EIntError do Handler(E.Message);
on E: EConvertError do Handler(E.Message);
end;
end;
end;

```

**فكرة:** لا يجب ان يكون ال Handler procedure متداخلاً في  
 procedure آخر فيمكن ان يتم تعريفه ك procedure . أو class  
 method يمكنك ان تفعل هذا، مثلاً، بحيث تستطيع ال procedure  
 المتعددة استدعاء نفس البرنامج المعالج.

### إعادة حدوث ال exception:

بينما تقوم بتطوير تطبيقك، فإنك تكتب العديد من ال procedure وال  
 functions التى توفر برامج معالجة exception، وقد تريد ان تضيف امكانات  
 جديدة لهذا ال code الموجودة فعلاً. على سبيل المثال، قد يكون لديك برنامجاً  
 يغلق ملفاً أثناء الحدوث لخطأ قرص. وقد تحتاج تركيبة اخرى لمعالجة نفس هذا ال  
 exception لاغراض خاصة بها. فبدلاً من اغلاق الملف فى مكانين مختلفين -  
 وهى عمل برمجى ضعيف يعقد الامور فى المستقبل - فيستطيع معالج ال  
 exception الثانوى ان يعيد حدوث ال exception، هذا يحافظ على exception  
 بحيث، بعد ان يؤدى البرنامج المعالج الثانوى اعماله، ينتقل ال exception الى  
 أعلى سلسلة الاستدعاء ليصل الى برامج معالجة اخرى تكون فى انتظاره.

## الباب التاسع عشر : التعامل مع ال exceptions

إذا كنت على دراية بالبرمجة المختصة بال objects فقد تفكر في إعادة exception كنوع من التقسيم الفرعى لل class فى وقت التشغيل . فقد يزيد البرنامج المعالج لل exception على البرامج المعالجة الأخرى بالامساك بأنواع معينة من ال exception ، لتنفيذ بعض الأعمال ، ومن ثم إعادة حدوث ال exception للابقاء عليه لمزيد من المعالجة . هذا يشبه الطريقة التى يستدعى بها method ما classes مشتقة method موروثة مما تفعله هذا ال code .

لمعالجة exception دون أن تدمر ال exception instance استخدم الكلمة الأساسية raise وحدها دون argument . افعل هذا مع كل انواع ال exception باستخدام code مثل التالية (يوضح التعليق اين تضع عبارة واحدة أو أكثر والتي قد تحدث ال exception الاصلى) .

```
try
{ Statements that might raise an exception }
except
ShowMessage('OOPs!');
raise; { Reraise the exception }
end;
```

إذا قامت أى عبارة فى ال try block بإحداث أى exception يقفز التنفيذ الى ال ShowMessage . بعد ذلك ، فإن استخدام ال raise وبدون argument يعيد حدوث ال exception . ان تشغيل هذا ال code يعرض رسالتين : الأولى بواسطة ال ShowMessage ، والثانية بواسطة برنامج معالجة ال exception الافتراضى للبرنامج (ولكن ، قد تكون بعض ال exception ولا تعرض أية رسائل -المزيد عن هذا فيما بعد) .

**تحذير:** تقوم ال code السابقة بالامساك بكل ال exception وهو أمر غير معقول بوجه عام -وقد يؤدي الى مشكلة كبيرة- مثلاً ، عدم معالجة حالة الخروج عن الذاكرة بشكل سليم . ولكن ، لان هذا ال code ايضاً يعيد حدوث كل ال exception ، فإن برامج معالجة ال exception الافتراضية لـ Delphi مازال لديها فرصة ان تتولى امر المشكلات الخطيرة . وفى حالات نادرة فقط عندما يكون لديك اسباباً جيدة لخرق هذه القاعدة ، استدع ال raise دائماً كما هو موضح فى ال except block الذى يمسك بكل ال exception .



لإعادة حدوث نوع معين من ال exception استخدم code مشابهة، ولكن إدخل raise. على سبيل المثال، فإن جزئية هذا ال raise:

```
try
{ Statements that might raise an exception }
except
on E: EOverflow do
begin
ShowMessage('OOPs!');
raise; { Reraise specific exception }
end;
end;
```

تمسك الحالات من ال EOverflow class، وتعرض رسالة، ثم تعيد حدوث نفس ال exception. و ال exception الأخرى التي تحدث في عبارات ال try block تنتهي ال procedure أو ال function على الفور، متخطية ال except block.

**ملحوظة:** حدوث exception ينهي على الفور ال procedure أو ال function التي تستدعي ال raise.

Note

## إنشاء Exception Classes:

عندما تحتاج الى حدوث exception، يمكنك اختيار واحدة من Delphi classes المشتقة من ال Exception (راجع القائمة السابقة) أو يمكنك إنشاء exception classes الخاصة بك. يمكنك ان تشتق classes الخاصة بك من ال Exception، أو يمكنك ان تنشئ ال Exception.

قم باشتقاق exception classes الخاصة بك من ال Exception اذا كنت تريد ان تقوم ببرامج المعالجة الافتراضية للتطبيق بالامساك بأى exception غير معالجة لل class. وبشكل عام، هذا هو افضل شئ تفعله. ولكن، يمكنك إنشاء class جديدة من تصميمك اذا لم تكن تريد لبرامج المعالجة الافتراضية ان تتعرف على exception.

**تحذيره** ان أى exception غير معالجة للـ classes ليست مشتقة من الـ Exception التى تؤدى الى خطأ تطبيق جسيم . يجب ان تعالج الـ Exception كـ حادث لـ exception غير مستندة على الـ Exception

Caution

.class

### :Custom exception classes

ان custom exception classes تعتبر نافعة فى تجميع معلومات حول مشكلات تحدث فى برنامجك . على سبيل المثال ، ان procedure رياضى يؤدى بعض انواع الحسابات قد يمرر قيم متغيرات غير مسموح بها فى شكل من custom exception class . فى الـ except block للبرنامج ، تستطيع عبارة ما ان تعرض هذه القيم .

هذا يعطى مستخدمى برنامجك معلومات يستطيعون استخدامها لاصلاح مشكلة-ادخال قيم صحيحة فى الـ display ، مثلاً . ولا احد يقدر رسالة خطأ مثل "File I/O error." اما رسالة مثل . "Filename contains the illegal character @" فتعطى مزيداً من المعلومات . يمكنك استخدام الـ exception custom class لانشاء هذا النوع من رسالة الخطأ . وتعتبر الـ customexception class غاية النفع ايضاً فى ازالة الاخطاء .

ان ابسط mothed لانشاء custom exception class هو بتحديد نوع class جديدة مستندة على الـ Exception class . على سبيل المثال ، ادخل تعريف الـ type التالى (يمكن ان تذهب فى التنفيذ ، ولكنك غالباً ما تريد ان تجعل الـ class متاحة لـ modules اخرى بحيث يستطيعوا انشاء ومعالجة exception للـ class الجديدة) :

type

```
TCustomException = class(Exception);
```

توفر الـ class الجديدة اسم class فريد لتعريف نوع معين من المشكلات-لا تحتاج الـ class الى هيكل . مع هذا التعريف ، يمكنك احداث exception للـ TCustomException class ، وتمسك بالاطء الخاصة بهذا النوع المعين . على سبيل المثال ، يمكنك ادخال هذه العبارة فى الـ Button OnClick :

```
raise TCustomException.Create('Custom exception');
```

عندما تقوم بتشغيل البرنامج وضغط الزر، تتلقى برامج معالجة ال exception الافتراضية ل Delphi حالة ال exception وتعرض الرسالة. وتستطيع أيضاً برامج ال exception الخاصة بك ان تمسك ال TCustomException objects باستخدام code مثل هذه:

```
try
{ Statements that might raise an exception }
except
on E: TCustomException do
    ShowMessage(E.Message);
end;
```

هذا يشبه الامثلة الاخرى الموجودة في هذا الباب، والفرق الوحيد هو ان ال code تمسك بال exception class TCustomException فقط والانواع الاخرى من ال exception تمر الى أعلى في سلسلة الاستدعاء حتى تجد معالج أو تصل الى برامج معالجة ال exception الافتراضية ل Delphi.

واحد الاسباب لانشاء ال exception class الخاصة بك هو لتخزين قيم توفر معلومات إضافية حول خطأ ما. على سبيل المثال، افترض انك تريد ان يقوم البرنامج بميكشاف ما اذا كان المستخدمون قد ضغطوا داخل تكوين جرافيك يقوم البرنامج برسمه. ولان التكوين لا يعد exception، فيجب ان تفحص كل ضغط الفأرة في نافذة ال form وتستجيب بشكل ملائم - تنهى التطبيق، مثلاً، فقط اذا كان المستخدم قد ضغط داخل تكوين معين. تستطيع بالطبع ان تقوم بهذا باستخدام Delphi components، ولكن هذه المشكلة توضح الاستخدام الجيد لل custom exceptions.

**على القرص المدمج:** يوضح تطبيق ال MouseExcept على القرص

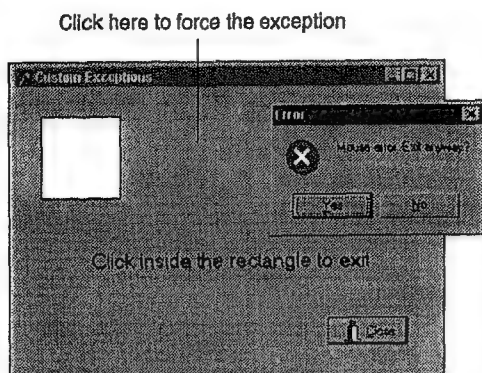
الدمج في دليل Source\MouseExcept كيفية إنشاء

استخدام exception. يوضح شكل (١٩-٩) عرض البرنامج توضح

القائمة (١٩-٤) ال source code للبرنامج.



## الباب التاسع عشر : التعامل مع الـ exceptions



شكل (٩-١٩): يوضح تطبيق الـ MouseExcept إنشاء واستخدام exception مخصصاً. اضغط داخل المربع الصغير لإنهاء البرنامج؛ اضغط في أي مكان آخر في النافذة لحدوث exception

القائمة (٩-١٩): MouseExcept\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics,  
Controls,  
Forms, Dialogs, StdCtrls, Buttons, ExtCtrls;
```

```
type
```

```
TMainForm = class(TForm)
```

```
BitBtn1: TBitBtn;
```

```
Label1: TLabel;
```

```
procedure FormMouseDown(Sender: TObject;
```

```
Button: TMouseButton; Shift: TShiftState; X, Y:  
Integer);
```

```
procedure FormPaint(Sender: TObject);
```

```
private
```

```
{ Private declarations }
```

```
procedure CheckMouseLocation(X, Y: Integer);
```

```
procedure ExitOnClick(X, Y: Integer);
```



```

public
{ Public declarations }
end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

const
  rLeft  = 25;
  rTop   = 25;
  rRight = 100;
  rBottom = 100;

type
  TMouseEvent = class(Exception)
    X, Y: Integer;
    constructor Create(const Msg: string; XX, YY: Integer);
  end;

constructor TMouseEvent.Create(const Msg: string;
  XX, YY: Integer);
begin
  X := XX; // Save X and Y values in object
  Y := YY;
  Message := // Create message string
    Msg + ' (X=' + IntToStr(X) + ', Y=' + IntToStr(Y) +
    ')';
end;

procedure TMainForm.CheckMouseLocation(X, Y: Integer);

```

## exceptions الباب التاسع عشر : التعامل مع الـ

```
begin
    if (X < rLeft) or (X > rRight) or
       (Y < rTop) or (Y > rBottom) then
        raise
        TMouseEvent.Create('Mouse location error',
        X, Y);
end;

procedure TMainForm.ExitOnClick(X, Y: Integer);
begin
    try
        CheckMouseLocation(X, Y); // Bad values raise
        exception
        Close;                // Exit the program
    except
        on TMouseEvent do
            begin
                if MessageDlg('Mouse error. Exit anyway?',
                    mtError, [mbYes, mbNo], 0) = mrYes
                then
                    Close
                else
                    raise;
                end;
            end;
    end;
end;

procedure TMainForm.FormMouseDown(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    ExitOnClick(X, Y);
end;

procedure TMainForm.FormPaint(Sender: TObject);
begin
    Canvas.Rectangle(rLeft, rTop, rRight, rBottom);
```

end;

end.

ان ال form OnPaint الرئيسية لبرنامج ال MouseExcept يرسم مربع صغير باستخدام قيم ثابتة معرفة في قطاع ال implementation عندما تضغط الفأرة في أى مكان في النافذة، يقوم ال FormMouseDown لل OnMouseDown بتلقى قيم إحداثيات ال x وال y الخاصة بمؤشر الفأرة. وهذا ال procedure يستدعى ال ExitOnClick، محرراً نفس هذه القيم x و y على انها arguments. إن ال ExitOnClick يعد public procedure قمت بإضافة لل form class وقد تم تنفيذه كما هو موضح في القائمة.

ينفذ ال ExitOnClick اثنين من العبارات في ال try block. أولاً، يقوم باستدعاء ثانى procedure مضافاً الى ال class، وهو ال CheckMouseLocation، والذي يختبر ما اذا كانت ال x وال y داخل المربع. ويقوم ال procedure ايضاً باستدعاء ال Close لانتهاء البرنامج. ولان ال CheckMouseLocation يحدث exception اذا ماتم إكتشاف أى مشكلات خاصة بال x وال y، فإن عبارة ال Close يتم تخطيطها اذا ما ضغط المستخدم خارج المربع. وفي هذا ال event، يقوم ال except block الخاص بال exception بعرض رسالة خطأ ويسأل عما اذا كان ينهى البرنامج بأية حال. لقد قمت ببرمجة ال procedure بهذه الطريقة بحيث، اذا ما اجبت بـ No على ال dialog الرسالة، يقوم ال except block باعادة حدوث ال exception. لذلك يوضح البرنامج الاختيارى كلاً من استجابة البرنامج لل exception المخصص ويوضح ما يحدث عندما تتلقى برامج معالجة ال exception الافتراضية لـ Delphi نفس هذا ال object.

ويتم تعريف هذا ال object على انه حالة من ال class يعرفها البرنامج نفسه كما يلي:

type

TMouseEvent = class(Exception)

X, Y: Integer;

## الباب التاسع عشر : التعامل مع ال exceptions

```
constructor Create(const Msg: string; XX, YY: Integer);  
end;
```

تعتبر ال TMouseEvent مشتقة من ال Exception class التابعة لـ Delphi. وبالإضافة الى العناصر الموروثة عن ال Exception، فإن ال TMouseEvent تضيف اعضاء من اعداد صحيحة x و y، وتعرف constructor، والذي يتم استدعائه لإنشاء class objects، ومحتويات ال constructor كمايلي :

```
constructor TMouseEvent.Create(const Msg: string;  
  XX, YY: Integer);  
begin  
  X := XX; // Save X and Y values in object  
  Y := YY;  
  Message := // Create message string  
  Msg + ' (X=' + IntToStr(X) + ', Y=' + IntToStr(Y) + ')';  
end;
```

ان اول عبارتين فى ال constructor يحفظ قيم ال parameters XX وال YY فى ال x وال y المعرفين فى ال TMouseEvent. ويتم تعيين رسالة تتضمن هاتين القيمتين لل Message string الموروث عن ال Exception.

ولكى تفهم كيف يتم استخدام ال exception class المخصص، انظر ال CheckMouseLocation procedure. اذا كانت ال X أو ال Y خارج المربع الذى يرسمه البرنامج، فإن هذا procedure يحدث exception object لل TMouseEvent class مستخدماً هذه العبارة:

.raise

```
TMouseEvent.Create('Mouse location error', X, Y);
```

هذا يؤدى الى إنشاء TMouseEvent، والذي يحفظ قيم ال X وال Y لموقع المؤشر المتحرك، وينهى ال CheckMouseLocation على الفور. وتعود الى ال ExitOnClick، اذا حدث هذا ال exception، يتم تخطي عبارة ال Close ويتولى ال except block التحكم.

بالرغم من ان هذا البرنامج النموذجى قد يبدو معقداً للغاية بحيث يصعب أداء عملياته ، فإن فى تطبيق أكثر اتساعاً ذا صور جرافيكية كثيرة ، تقوم مثل ال class TMouseEvent بتبسيط ال code الى حد كبير . وبدون ال exception class المخصصة ، فقد يستخدم البرنامج العديد من عبارات if الشرطية التى يصعب متابعتها لاختبار مواقع الفأرة ، مما يجعل إزالة الاخطاء امراً عسيراً . ان تركيز كل معالجة الاخطاء فى exception class مخصصة ، وباستخدام ال try-except blocks لاكتشاف الاخطاء ، يحافظ على ال code ويحسن من قدرتها .

### exception classes الغير مشتقة:

يمكنك ايضاً انشاء exception classes جديدة تماماً غير مستندة على ال Exception . وابسط طريقة class ليس لها هيكل . ولكن لأنها Class اساسية فيجب ان ينتهى تعريفها بالكلمة الاساسية end وفصلة منقوطة . وهذا هو احد الأماكن النادرة فى ال Object Pascal تكون فيه ال end غير مطلوبة ليكون هناك body . على سبيل المثال ، ابدأ تطبيقاً جديداً وإدخل تعريف ال type هذا فى قطاع ال implementation :

```
type
  TBareException = class
  end;
```

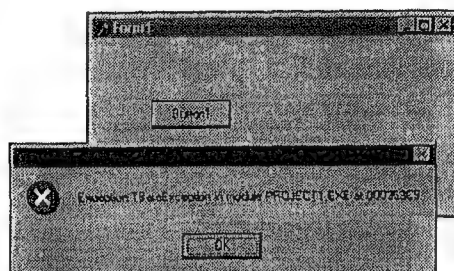
بعد ذلك ، أضف Button على ال form ، وقم بانشاءOnClick خالى وبضغط هذا ال event مرتين فى نافذة ال Object Inspector . قم ببرمجة ال event handler كما يلى :

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  raise TBareException.Create;
end;
```

ان استدعاء ال Create method الخاص بال claas يؤدى الى إنشاء object من ال class ، والذى يتم حدوثه ك exception . عندما تقوم بالcompile وتشغيل البرنامج بضغط F9 ، فإن نقر الزر ينفذ هذا ال code ويحدث ال

## الباب التاسع عشر : التعامل مع ال exceptions

exception . ويتلقى ال exception handler الافتراضى ل Delphi فى هذا ال object ، ويعرض ال message dialog الموجود فى شكل (١٩-١٠).



شكل (١٩-١٠): exception handler الافتراضى ل Delphi يعرض هذه الرسالة لـ exception object من class ليست مشتقة من ال Exception

**ملحوظة:** لان كل ال class تكون مشتقة من ال TObject ، فحتى ال class الاساسية الجديدة تأتى مع ال Create وال Free methods المقامة بالفعل .




إذا استخدمت هذه التقنية ، فمن الافضل ان تعالج كل حالات ال exception classes الخاصة بك ، وقد تريد ايضاً ان تقوم بتثبيت برنامج معالجة ال exception الافتراضى الخاص بك كما هو موضح مؤخراً فى هذا الباب . يمكن لل class ان تعرف methods ومتغيرات اضافية وما تصنعه فى ال exception classes المخصصة التابعة لك أمر يرجع إليك .

### Exception base class

يجعل ال exception classes الخاصة بك ترث ال Exception classes ، فإن ال classes ترث ثروة من الاعضاء . على سبيل المثال ، تقدم ال Exception class عدد من ال methods لإنشاء exception class بطرق مختلفة . وكل هؤلاء ال constructors يحملون اسماء تبدأ بال Create ، وكلهم لديهم نفس الهدف الوحيد وهو إنشاء string رسالة خطأ من انواع مختلفة من ال arguments . يمكنك استدعاء هؤلاء constructors لإنشاء exception instances من ال class المشتقة . أو يمكنك استدعائهم لـ Exception class أو لأى من ال classes المشتقة المذكورة آنفاً فى هذا الباب . سوف أمر على (constructors) التابعين ل class لـ Exception باختصار هنا-ومن المؤكد ان يفى احدهم باحتياجاتك .

وابسط هؤلاء هو الـ Create الذى رأيت به بالفعل . قم بتمرير ثابت أو متغير string كرسالة خطأ :

```
raise Exception.Create('Trouble in Paradise');
```

**فكرة:** ادخل العبارة السابقة فى الـ Button OnClick، واضغط الزر  لترى النتائج. افعل نفس الشيء مع الامثلة الاخرى المذكورة فى نفس هذا الفصل.

لإنشاء رسالة تحتوى على معلومات إضافية، قيم متغيرة، مثلاً، استدع constructor البديل CreateFmt. والى (constructors)، قم بتمرير string مع اوامر معينة زائد مجموعة من القيم ليتم ادخالها فى الـ string. (للمعاونة فى هذه التقنيات انظر (استخدام دالة الـ Format فى الباب السابع، وكذلك انظر الـ Format فى الـ Delphi online help). على سبيل المثال، تقوم هذه العبارة بإحداث exception مع رسالة خطأ تعرض إثنين من القيم الصحيحة :

```
raise Exception.CreateFmt('Error: X=%d Y=%d', [X, Y]);
```

ونتيجة لهذه العبارة، يرى المستخدمون رسالة خطأ مثل التالية :

```
Error: X=-1 Y=12
```

عند انشاء exception objects بهذه الطريقة، فقد تريد استخدام مجموعة error message resource strings فى الـ SysConst unit الخاصة بـ Delphi. يذكر الجدول (١٩-١) الثوابت الموجودة فى هذا الجدول ويوضح الـ strings المرتبطة بها. إنها متصلة بكل التطبيقات، لذا يمكنك استخدامها بدلاً من تعريف الـ strings الشبيهة الخاصة بك لاستخدام الـ strings اضعف تعريف الـ uses الى قطاع . الـ unit implementation :

```
uses
```

```
SysConst;
```

بعد ذلك، قم بإحداث exception باستخدام واحداً من الثوابت المعروفة :

```
raise Exception.Create(SDiskFull);
```

## الباب التاسع عشر : التعامل مع ال exceptions

**ملحوظة:** ان النسخ الاولى من Delphi قد عرفت في ال SysUtils unit الثوابت الموجودة في جدول (١٩-١) كمعرف في اعداد صحيحة من ال resource ، والتي يمكنك تمريرها الى ال CreateRes constructor Exception class . وهذه الثوابت قد تم تحديدها الآن كـ resource strings في ال SysConst unit . ولكن مازالت ال CreateRes متاحة لإنشاء ال Exception objects التي تقوم بتحميل error message strings من جدول ال Windows string resource-ولكن هذا ال method يعتبر غير ملائم لأغلب تطبيقات Delphi .

Note

جدول (١٩-١) : ال SysConst Unit تعرف رسائل الخطأ المعيارية هذه، والتي يمكنك استخدامها لإنشاء ال Exception Class الخاصة بك

Identifier	String
SinvalidInteger	'"%s" is not a valid integer value';
SinvalidFloat	'"%s" is not a valid floating point value';
SinvalidDate	'"%s" is not a valid date';
SinvalidTime	'"%s" is not a valid time';
SinvalidDateTime	'"%s" is not a valid date and time';
StimeEncodeError	'Invalid argument to time encode';
SdateEncodeError	'Invalid argument to date encode';
SOutOfMemory	'Out of memory';
SinOutError	'I/O error %d';
SfileNotFound	'File not found';
Identifier	String
SinvalidFilename	'Invalid filename';
StooManyOpenFile	'Too many open files';
SAccessDenied	'File access denied';
SEndOfFile	'Read beyond end of file';
SDiskFull	'Disk full';
SinvalidInput	'Invalid numeric input';
SDivByZero	'Division by zero';



SRangeError	'Range check error';
SintOverflow	'Integer overflow';
SInvalidOp	'Invalid floating point operation';
SZeroDivide	'Floating point division by zero';
SOverflow	'Floating point overflow';
SUnderflow	'Floating point underflow';
SInvalidPointer	'Invalid pointer operation';
SInvalidCast	'Invalid class typecast';
SAccessViolation	'Access violation at address %p. %s of address %p';
SStackOverflow	'Stack overflow';
SControlC	'Control-C hit';
SPrivilege	'Privileged instruction';
SOperationAborted	'Operation aborted';
SException	'Exception %s in module %s at %p. #\$0A' %s %s';
SExceptTitle	'Application Error';
SInvalidFormat	Format "%s" invalid or incompatible with argument';
SArgumentMissing	'No argument for format "%s"';
SInvalidVarCast	'Invalid variant type conversion';
SInvalidVarOp	'Invalid variant operation';
Identifier	String
SDispatchError	'Variant method calls not supported';
SReadAccess	'Read';
SWriteAccess	'Write';
SResultTooLong	'Format result longer than 4096 characters';
SFormatTooLong	'Format string too long';
SVarArrayCreate	'Error creating variant array';
SVarNotArray	'Variant is not an array';

## الباب التاسع عشر : التعامل مع ال exceptions

SVarArrayBounds	'Variant array index out of bounds';
SExternalException	'External exception %x';
SAssertionFailed	'Assertion failed';
SIntfCastError	'Interface not supported';
SAssertError	'%s (%s, line %d)';
SAbstractError	'Abstract Error';
SModuleAccessViolation	Access violation at address %p in module "%s". %s of address %p';
SCannotReadPackageInfo	'Cannot access package information for package "%s"';
sErrorLoadingPackage	'Can't load package %s.'#13#10'%s';
SInvalidPackageFile	'Invalid package file "%s"';
SInvalidPackageHandle	'Invalid package handle';
SDuplicatePackageUnit	'Cannot load package "%s." It contains unit "%s," + 'which is also contained in package "%s"';
SWin32Error	'Win32 Error. Code: %d.'#10'%s';
SUnkWin32Error	'A Win32 API function failed';
SNL	'Application is not licensed to use this feature';

والمتبقى من ال Exception constructors يعتبر صور من الاهداف السابقة . فيما يلي نماذج منهم . يقوم ال CreateResFmt بتحميل string-table resource مع امر string format (%s) (لاحظ انه يجب عليك تمرير مجموعة string الى هذا ال constructor) . يضيف ال CreateHelp ال help-context المعرف (واستخدامه امر يرجع الى ال exception handler الخاص بك) . ال CreateFmtHelp يجمع string ال forming مع ال help-context المعرف . وال CreateResHelp يجمع string-table resource مع ال help-context المعرف . ويضع ال CreateResHelp كل هذه الاشياء مع ال string resource والذي يحتوى على forming codes مثل ال %s أو %d ومعامل : help-context

```
raise Exception.CreateResFmt(SInvalidInteger, [IntToStr(N)]);
raise Exception.CreateHelp('Whoops, sorry', 123);
raise Exception.CreateFmtHelp('Error N=%d', [N], 123);
raise Exception.CreateResHelp(SFileNotFound, 123);
raise Exception.CreateResFmtHelp(SInvalidInteger, [S], 123);
```

## تقنيات Exceptional أخرى:

في الفصول التالية سوف نشرح تقنيات exception متقدمة تكون مفيدة في الظروف الخاصة.

### ، Silent exceptions

ان ال Silent exceptions هو exceptions ليس له تأثير ظاهر على البرنامج . ولكنه مفيد في إفشال العمليات المتداخلة بعمق . ان silent exception instance هي object من ال EAbort class ، والمشتقة من ال Exception class . وان برنامج معالجة ال exception الافتراضى لـ Delphi قد تم برمجته على الا يعرض أى رسائل خطأ عندما يتلقى ال EAbort exception . بدلاً من ذلك ، يقوم البرنامج المعالج الافتراضى بتدمير exception instance ويعيد التحكم الى البرنامج .

قم باستدعاء ال Abort procedure لاجداث exception من ال EAbort class . على سبيل المثال ، ان ال while loop التى تفحص ال NormalCondition Boolean flag . يمكنها ايضاً ان تفحص flag آخر ، وهو ال SpecialConditionFlag ، وتخرج من ال loop باستدعاء ال Abort :  
while NormalCondition do  
begin  
if SpecialConditionFlag = False then  
Abort;  
end;

ان استدعاء ال Abort يعتبر مساوياً لإحداث ال EAbort exception class ، والذي قد تقوم به بالطريقة الصعبة لتمرير رسالة مع ال object (ولكن لان ال exception يكون صامتاً ، فلا يعرض البرنامج ابداً هذا ال string) :

```
raise EAbort.Create('Operation aborted');
```

ان اهمية هذه التقنية هي انها لا تنتج أى تأثير يظهر على الشاشة . تنتهى ال loop بهدوء بأحداث exception من ال EAbort class ، والذي يتلقاه برنامج معالجة exception الافتراضية ويعيد التحكم الى البرنامج . ان تمرير string بهذه الطريقة فى silent exception يمكن ان يكون مفيداً فى عملية (debugging) على سبيل المثال ، اذا كان برنامجك يحتوى على exception handler ، فإن هذا البرنامج المعالج قد يستجيب للEAbort كما يفعل مع أى exception اخرى . وبالتبادل اذا كنت لا تريده ان يتعرف على ال silent exception (لذا فهى تبقى صامتة) ، استخدم ال try-except block التالى ، والذي يعالج ال exception ولكن يستدعى ال ShowMessage فقط اذا لم يكن ال exception من ال EAbort :

```
try
{ Statements that may call Abort }
except
on E: Exception do
if not (E is EAbort) then
ShowMessage(E.Message);
raise;
end;
```

### استبدال برنامج معالجة ال exception الافتراضى:

للحصول على عمل متقدم بحق ، يمكنك ان تبدأ فى البحث عن برنامج معالجة exception افتراضى مخصص يعمل على مستوى ال Application وهذه التقنية توفر وصولاً الى كل ال exception التى لم يتم معالجتها ، وهو ما قد يكون مفيداً فى إزالة اخطاء امكانات ال exception handler الخاصة بالتطبيق وكذلك استبدال dialog box برنامج المعالجة الافتراضى مع المزيد من الوصف والايضاح .

حتى اذا لم تحتاج الى كتابة exception handler افتراضى لتطبيقك ، فقد تريد ان تجرب التقنية الموجودة فى هذا الفصل . وان تكون قادراً على الامساك بال exception التى لم يتم معالجتها على مستوى التطبيق هو أمر غاية فى القيمة وذلك

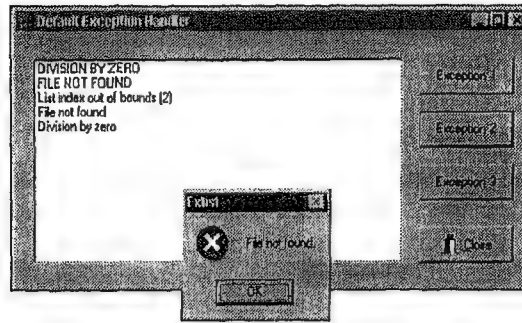
لإزالة أخطاء برنامجك . ان التطبيق بهذا الفصل يستخدم هذا ال method للحفاظ على قائمة بكل ال exception التي لم يتم معالجتها اثناء البرنامج . وهذه القائمة ذات قيمة عالية في تتبع مصدر المشكلات الصعبة .

ان ال TApplication يوفر procedure ، وهو ال HandleException ، والذي يقوم بمعالجة أى exceptions لم يتم معالجتها والتي تم حدوثها اثناء تنفيذ البرنامج . لا يجب عليك ان تحاول تجاهل هذا ال method- فإن هذا سوف يتجنب تماماً برامج معالجة ال exceptions الافتراضية لـ Delphi ، ومن المؤكد ان يؤدي الى مشكلة . بدلاً من ذلك ، لزيادة برنامج المعالجة الافتراضى ، يمكنك توفير procedure للـ Application OnException event ان ال Handle Exception method يستدعى OnException procedure ، اذا تم تعيين واحداً ، بدلاً من عرض رسالة خطأ افتراضية .

**على القرص المدمج:** يوضح شكل (١٩-١١) تنفيذ تطبيق ال ExList . تمر كل ال exception التي لم يتم معالجتها من خلال ال handler الجديد ، والذي يضيف فى هذا البرنامج رسائل الخطأ الى ال ListBox . هذا يعطى تتبعاً تاماً لل exception التي لم يتم معالجتها فى التطبيق يوجد البرنامج على القرص المدمج فى دليل ال Source\ExList . قم بتشغيل هذا البرنامج واضغط ازرار ال Exception الثلاثة لتوليد ثلاثة انواع من ال exception . بعد ان تغلق dialog box الافتراضى الناتج ، يضيف البرنامج رسائل خطأ .



أن البرنامج مضيف رسائل ال exception error الى ال ListBoxform . ان ضغط ال Exception1 يحدث exception القسم على صفر (أو divide-by-zero) . ان ضغط ال Exception2 يحدث exception عدم العثور على الملف (أو file-not-found) ان ضغط ال Exception3 يحول كل ال strings المذكور الى حروف كبيرة- ولكن البرمجة تحتوى على خطأ يحدث index-out-of-bounds exception . توضح القائمة (١٩-١٥) ال source code للبرنامج .



شكل (١٩-١١)، تطبيق الـ ExList يوضح كيفية كتابة برنامج exception handler افتراضى

القائمة (١٩-١٥) ExList\Main.pas:

unit Main;

interface

uses

Windows, SysUtils, Messages, Classes, Graphics,  
Controls,  
Forms, Dialogs, Buttons, StdCtrls;

type

TMainForm = class(TForm)

Listbox1: TListBox;

Button1: TButton;

Button2: TButton;

Button3: TButton;

BitBtn1: TBitBtn;

procedure FormCreate(Sender: TObject);

procedure Button1Click(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure Button3Click(Sender: TObject);

private

{ Private declarations }

public

```

{ 1. Declare the OnException event handler }
procedure NewOnException(Sender: TObject;
    E: Exception);
    { Public declarations }
end;

```

```

var
    MainForm: TMainForm;

```

```

implementation

```

```

{$R *.DFM}

```

```

{ 2. Implement the OnException event handler }
procedure TMainForm.NewOnException(Sender: TObject;
    E: Exception);
begin
    Application.ShowException(E);
    ListBox1.Items.Add(E.Message);
end;

```

```

{ 3. Assign the event handler to Application.OnException }
procedure TMainForm.FormCreate(Sender: TObject);
begin
    Application.OnException := NewOnException;
end;

```

```

{ Raise a divide-by-zero exception }
procedure TMainForm.Button1Click(Sender: TObject);
var
    I, J, K: Integer;
begin
    I := 0;
    J := 10;
    try
        K := J div I; { Divide by zero! }
    end;
end;

```

## exceptions الباب التاسع عشر : التعامل مع الـ exceptions

```
=====
{ The following statement doesn't execute, but it is
  needed so the optimizer in Object Pascal, which
  notices
    that K isn't used in this procedure, doesn't strip
    out the preceding statement. Smart compiler. }
  ShowMessage('K=' + IntToStr(K));
except
raise;
end;
end;

{ Raise a file-not-found exception }
procedure TMainForm.Button2Click(Sender: TObject);
var
  T: TextFile;
begin
  AssignFile(T, 'XXXX.$$$');
  Reset(T); { Open a non-existent file! }
  try
    { You would normally use T here }
  finally
    CloseFile(T); { For safety's sake }
  end;
end;

{ Raise an index-out-of-bounds exception }
procedure TMainForm.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  with ListBox1.Items do
    for I := 0 to Count do { Should be Count - 1! }
      Strings[I] := Uppercase(Strings[I]);
end;

end.
```



ان ال Application event handle يتطلب عملاً أكثر من إنشاء واحداً، فلنقل مثلاً ال Button OnClick. وهذا لان ال Application object لا يمكن ان يتوصل إليه ال Object Inspector الخاص بـ Delphi's. لذلك فإن إنشاء OnException يتطلب ثلاث خطوات، والمرقمة في التعليقات الموجودة بالقائمة (وهذه التعليقات مكتوبة بخط سميك ليسهل العثور عليها) وهذه الخطوات هي:

١- قم بتعريف OnException في ال form class. يجب ان يكون له ال parameters الموضحة في القائمة، بالرغم من ان تحديد اسم ال procedure يرجع إليك.

٢- قم بتنفيذ OnException. تذكر ان تسبق اسمه باسم ال form class نقطة.

٣- قم بتعيين ال event handler لل Application.OnException. ان ال HandleException method الخاص بال Application object يتأكد مما اذا كان البرنامج قد قام بتعيين ال procedure لل OnException. اذا كان كذلك، يقوم ال HandleException باستدعاء ال procedure لأي exceptions يتلقاه لم يتم معالجته.

**ملحوظة:** ان ال custom OnException event handler يتلقى ال exceptions المشتقة فقط من ال Exception class فيما عدا ال EAbort exceptions التي تبقى صامته.

Note

يمكنك اداء أى اعمال تريدها في ال OnException event handler. في البرنامج [انظر الخطوة رقم (٢)] تقوم العبارة الاولى باستدعاء ال ShowException method الخاص بال Application. وهذا ما يقوم به البرنامج الافتراضى بالفعل. بالإضافة الى ذلك، يستخدم البرنامج ال Add method لقائمة ال Items string الخاصة بال ListBox1 لحفظ نسخة من كل رسالة خطأ لاحظ ان ال event handler يتلقى ال exception object على انه parameter E. (هذا ال object يتم تدميره فى أى مكان آخر، فلا يجب عليك تحريره).

## الباب التاسع عشر : التعامل مع ال exceptions

يمكنك ان تجعل ال event handler الافتراضى صامتاً بحذف الاستدعاء لل Application ShowException . ولكن لا تفعل هذا دون تمييز ، فقد يخفى عيب خطير فى ال code ان OnException الخاص بك يجب ان يعرض نوع من الرسائل عند تلقى exception لم يتم معالجتها .

### افكار للمستخدم الخبير

● يجب أن تفكر فى ال exception ، كأنما معدة لاعادة انواع متعددة من ال object من ال procedures وال functions . من الناحية الفنية ، يمكنك ان تحدث exception لتمرر objects وتستخدم ال try-except blocks للامساك بهذه ال objects . إننى لا ارشح لك ان تستخدم ال exception بهذه الطريقة . ولكن من المفيد ان تدرك ان استخدامها للتحكم فى الخطأ هو امر ضرورى . ان معنى ال exceptions objects ، خاصة تلك التابعة ل classes خاصة بك ، يرجع إليك لتحديد تعريفه .

● ان ال HandleException method الافتراضى فى ال TApplication يرسل رسالة ال wm\_CancelMode الخاصة بال Windows لإطلاق الفأرة اذا كانت مقيدة فى وقت حدوث ال exceptions . وهذا ايضا يؤدي الى إغلاق list boxes ، وال combo boxes ، و drop-down lists ، والقوائم .

● عند إنشاء أى اشتقاق ل class من ال exception classes ، فضع بحذر عبارات on-do والتي تشير الى ال class لمعالجة exception objects لأغلب ال classes المشتقة من ال Exception أولاً . فكر فى ال code التالية ، والتي تشبه الامثلة الاخرى فى هذا الباب (ان عبارة ال ShowMessage تستخدم ال K لذا فإن ال optimizer الخاص بال Object Pascal لا يحذف عبارة القسمة السابقة) :

try

```
K := J div I;
```

```
ShowMessage('K=' + IntToStr(K));
```

```
except
```

```
on E: EDivByZero do // This must come first!
```

```

ShowMessage('Divide error');
on E: EIntError do // Trap all other EIntErrors here
ShowMessage('Other integer math error');
end;

```

• تستخدم هذه الـ code لاثنين من عبارات الـ on-do (يمكن ان يكون لديك أى عدد فى الـ except block). تمسك العبارة الاولى الـ EDivByZero exceptions، ربما لتفعل شيئاً خاصاً مع هذه الانواع من الاخطاء. وتمسك العبارة الثانية بكل الـ exception المشتقة من الـ EIntError class، والتي تعتبر ايضاً الـ class الاساسية للـ EDivByZero. وسيكون من الخطأ ان تعكس عبارات الـ on-do لأنه اذا كانت الـ exception للـ class الام قد تم الامساك بها اولاً، فإنها سوف تشمل الـ exceptions EDivByZero (والمشتقة من الـ Exception) ولن يتم تنفيذ code خطأ الـ Divide الخاصة.

• ان تطبيق Delphi له إثنين من برامج معالجة الـ exception الافتراضية. يتم توفير واحد منها بواسطة الـ VCL ويحمى كل procedure تمر من خلاله الرسالة ويمسك هذا البرنامج المعالج بكل الـ exception التي لم يتم معالجتها، ويعرض رسالة خطأ، ويستمر فى تنفيذ البرنامج. اما الـ exception handler الافتراضى الآخر فموجود فى الـ RTL (مكتبة وقت التشغيل)، وتوفره الـ SysUtils unit. ويأتى هذا البرنامج المعالج فى مكانة اقل من البرنامج المعالج الافتراضى فى الـ VCL، لذا يندران نراه عاملاً. عندما يمسك هذا البرنامج المعالج الـ exception، فإنه يعرض رسالة مفصلة (مع معلومات عنوان الـ exception لإزالة الخطأ)، ثم ينهى التطبيق.

### المشروعات التى يمكنك تجربتها

(١٩-١): اعد كتابة القائمة (١٩-١٣) لاستدعاء function تحول خاصية الـ Edit1 Text الى قيمة عدد صحيح. هذا يوضح كيف يمكنك، مع الـ exceptions، ان تقوم بتعديلات كبيرة لبرنامج وتظل واثقاً من منطق معالجة الاخطاء الخاص بك.

## الباب التاسع عشر : التعامل مع الـ exceptions

(١٩-٢) : استبدال برنامج معالجة الـ exceptions الافتراضي للـ Application بالنسخة الخاصة بك التي تعرض dialog box اكثر معاونة . على سبيل المثال ، قد يحتوى الـ dialog الخاص بك على زر Help يمكن للمستخدمين ان يضغطوه للحصول على معلومات حول انواع معينة الـ exceptions .

(١٩-٣) : قم بتحويل تطبيق الـ ExList الى تركيبة يمكنك دمجها فى أى تطبيق لإزالة اخطاء الـ exceptions التي لم يتم معالجتها . اصف امراً الى الـ module الخاصة بك لحفظ قائمة رسائل الـ exceptions فى ملف نص .

### ملخص:

- ان المطورين ذوى الخبرة يقومون بدمج code معالجة الخطأ فى برامجهم اثناء كتابتها . ان آليات الـ exception-handling الخاصة بـ Delphi تجعل هذا الامر سهلاً وبسيطاً .
- يمكن ان تحدث الـ expressions من عدة مصادر . تستطيع الـ Components ، والتعبيرات الرياضية ، وعمليات الادخال/ والاخراج من الملف ، وقصور الذاكرة واحداث اخرى تحدث الـ expressions . يمكنك ايضاً ان تحدث الـ expressions لتخبر تطبيقك عن مشكلة ما .
- ان الـ Protected-statement blocks تستخدم الكلمات الاساسية الـ try-except لمعالجة الـ exceptions قد تحدثها العبارات الموجودة فى جزئية الـ try .
- تستخدم الـ Protected-resource block الكلمات الاساسية الـ try-finally لتنفيذ code إجبارية مثل تحرير ذاكرة مخصصة اذا حدث الـ exceptions لعبارة فى الـ try block .
- ان الـ exception هو object من class ، تكون مشتقة غالباً من الـ class الـ Exception ، تصف طبيعة الـ exception عند حدوث الـ exception . يؤدي الى إنشاء هذا الـ object ومعالجة الـ exception تؤدي الى تدميره . لا يجب ان تحرر الـ exception instance ما بوضوح .

● استخدام عبارات ال on-do فى ال except block لمعالجة انواع معينة من ال exception عند استخدام هذه التقنية، تذكر ان أى exception لم يتم معالجتها من انواع غير مذكورة فى عبارات ال on-do تؤدى الى إنهاء ال procedure أو ال function الحالية بعد ال except block. استخدم ال try-finally blocks لمسح أى resource مخصصة فى مثل هذه الحالات.

● لاجداث exception استدع raise مع exception instance تم إنشاؤها حديثاً. لإعادة حدوث exception بعد معالجته فى ال except block، استدع raise بلا arguments.

● يمكنك انشاء exception classes خاصه بك، إما من لاشىء، أو باشتقاقها من ال Exception. فى اغلب الحالات، يجب ان تشتق exception classes الخاصة بك من ال exception classes لان برامج معالجة ال exception من هذه الانواع فقط.

● توفر ال Exception class عدداً من ال Create constructors لانشاء رسائل الخطأ من strings وقيم string-table resources.

● قم باستدعاء ال Abort لأحداث exception صامت من ال EAbort class. ان برامج معالجة ال exception الافتراضية لا تعرض dialog رسالة خطأ لل EAbort exceptions.

● قم بزيادة برنامج معالجة ال exception الافتراضى وذلك بتعريف وتنفيذ وتعيين ال OnException فى ال class المشتقة من ال TForm والخاصة ببرنامجك. انظر تطبيق ال ExList على القرص المدمج لمعرفة التعليمات استخدم هذه التقنية لإزالة اخطاء ال exception التى لم يتم معالجتها ببرنامجك، أو لاستبدال dialog رسالة الخطأ الافتراضى.

يقدم الباب القادم افكاراً وتقنيات اساسية للقراء المهتمين بكتابة  
ActiveX controls components.

\*\*\*\*\*

# الباب العشرون

## إنشاء custom Components

### محتويات هذا الباب:

- Components
- معلومات حول الـ Components
- معلومات حول الـ packages
- تطوير الـ Components
- فهم الـ component design
- إنشاء الـ ActiveX controls

عندما تكتسب مزيداً من مهارات Delphi، فقد تبدأ تتسأل كيف تعمل الـ components في الحقيقة. أو، تكون لديك فكرة جيدة عن محاور نشاط يصنع component رائع ان اغلب مطوري Delphi لا يحتاجون الى بناء components، ولكن اذا كنت مهتم بهذا الموضوع المتقدم، فإن هذا الباب يساعدك على البدء في تصميم component.

في الفصول التالية، سوف تتعرف على أجزاء الـ component وكيف تتفاعل مع بيئة تطوير Delphi حيث يمكنك إنشاء، اختبار وتثبيت components جديدة في لوحة Delphi components، ويمكنك إلقاء نظرة على العديد من تقنيات إنشاء الـ components الأساسية. يمكنك أيضاً تحويل احد الـ components في هذا الباب الى ActiveX control. بعد قراءة هذا الباب، سوف تكون مستعداً لان تذهب الى ما وراء الاساسيات في تصميم الـ component وهو موضوع يمكن ان يملأ كتاباً بهذا الحجم.

## معلومات حول الـ components:

ان الـ components هو Object Pascal class التى تلتزم ببعض القواعد واللوائح . يمكنك بناء components من component موجوداً بالفعل ، ويمكنك إنشاء component من لاشئ . ولكن فى الغالب تقوم بتأسيس الـ component على تلك الـ component التى تأتى مع Delphi . على سبيل المثال ، اذا اردت ان تضيف بعض التحسينات الى زر ، يمكنك ان تؤسس component جديداً على الـ TButton class التابعة لـ Delphi وتضيف أى اشيء جديدة تريدها .

فيمايلى بعض الاسباب التى قد تجعلك تريد إنشاء components . هذه ليست قواعد جامدة- إنها مجرد اقتراحات قد تساعدك فى ان تقرر ما اذا كنت تريد إنشاء components .

• إنك تحتاج الى تعديل components موجود بالفعل لغرض خاص-على سبيل المثال ، تريد Image components يستطيع ان يقوم بتحميل bitmap من الـ hardware التى تطورها شركتك . بتأسيس الـ component على الـ Image تكون كل خصائص و methods و events هذا الـ components متاحة فى تصميمك . يمكنك ان تشتق components جديدة من الـ TListBox ، TButton ، أو أى component class اخرى . هذا يؤدى ايضاً الى استخدام جيد للميراث المختص بال-object . فإن الـ class الجديدة ترث ، إمكانيات الـ class الأساسى بدلاً من نسخ أو تعديل code موجودة بالفعل . ان الوراثة تساعدك على تنظيم مكتبات الـ components بشكل افضل ، وكذلك تسهل الحفظ

• إنك تريد انشاء واجهة تطبيق Delphi component للـ control المخصص للـ Windows . عادة يمكنك ان تشتق هذا النوع من الـ TWinControl ، والتى توفر الخصائص والـ methods والـ events الاساسية المشتركة فى كل الـ Windows controls .

• إنك تريد ان تنشئ component جديد تماماً من لاشئ هذا قد يحتاج الى بعض الجهد ، ولكن قد توفر على نفسك كثير من الجهد والوقت بتأسيس الـ components على الـ TCustomControl ، والتى توفر GDI Canvas مع الخصائص والـ methods والـ events العامة فى كل الـ components .

## الباب العشرون: إنشاء custom Components

• ان يكون لتطبيقك متطلبات جرافية خاصة ، وتريد ان تنشئ components يوفر هذه الامكانيات . ان اغلب components الجرافيك تستند على الـ TGraphicControl class . وهناك method فى هذا الباب تستخدم هذا الـ method لإنشاء BarChart الذى يعرض BarChart من مجموعة من نقاط البيانات التى تريدها string list .

• إنك تريد ان تنشئ nonvisual component واحداً لا يظهر على الـ component palette . لقد قابلت بعض حالات من الـ nonvisual components - الـ TClipboard class مثلاً . إنها نادرة نسبياً ، ولكنها تستفيد من تقنيات إنشاء الـ components والتى تساهم فى قيمة البرنامج وسهولة الحفظ .

### مؤلفى الـ components ومستخدميها:

عبر هذا الكتاب حتى الآن ، كنت انت مستخدماً للـ component . وحين تنتهى من هذا الباب ، ستكون مؤلفاً للـ component . ان مستخدمى الـ component ينشئون objects متعددة من الـ components فى وقت التصميم بإدخالها فى نافذة from ، وفى وقت التشغيل باستدعاء الـ Create methods الخاصة بها يقوم مؤلفى الـ components بإنشاء components classes جديدة والتى يستطيع المستخدمون ان ينشئوا منها واحداً أو أكثر من الـ objects فى وقت التصميم وفى وقت التشغيل .

وهناك اختلافاً آخر بين مستخدمى الـ component ومؤلفى الـ component يختص بالوصول الى عناصر الـ component . كمستخدم component ، فإن لك حقوقاً مقيدة فى الوصول الى لب الـ component . على سبيل المثال ، يمكنك قراءة وكتابة الخصائص المنشورة فقط ، ويمكنك إنشاء events فقط لتلك الاعمال التى يتحها الـ component فى نافذة الـ Object Inspector .

كمؤلف للـ component ، فليس لديك هذه القيود . إنك تقرر أى الـ component تنشرها وكيف تنشئ الاعمال التى يحتاجها الـ component . ان كل عناصر الـ component متاحة لبرنامجك (فيما عدا الاعضاء الخاصة فى الـ classes الاساسية ، والتى كما تعرف ، يمكن التوصل إليها فقط فى هذه الـ classes) ، كما يمكنك الوصول الى العناصر المحمية فى الـ component الموجودة بالفعل . وفى



مقابل هذه الحرية، فإنك تقبل مسئولية كبيرة فى كتابة code قوية. ان الاخطاء على مستوى ال component يكون لها عواقب وخيمة أكثر منها فى اخطاء التطبيق.

ان إنشاء components خاصة بك :

- يوفر وصولاً الى الاجزاء المقيدة فى ال components.
- يجعل من الممكن اضافة وتعديل خصائص وethods وevents ل components موجودة بالفعل.
- يستلزم معرفة عملية جيدة بال Object Inspector، والبرمجة المختصة بال object، ومعالجة ال exception.

لجعل هدفك هو ان يكون ال component أمته الاستخدام حتى من قبل المبتدئين. اذا تطلب component ما عنصراً معيناً، فيجب ان يوفر قيمة افتراضية فى حالة اذا لم يقدم المستخدمون قيمة معينة. لا تفترض ان المستخدمين يعرفون كيف يقومون بنشر ال component بشكل سليم. من الناحية النموذجية، يجب ان يكون المستخدمين قادرين على ان يفعلوا أى شئ بال components دون احداث مشكلة خطيرة.

### بعض المصطلحات النافعة:

فيما يلى تعريفات للمصطلحات التى تحتاج ان تعرفها لإنشاء components. ان القاموس يلتزم باستخدام هذه المصطلحات فى المعلومات الاخرى المنشورة عن Delphi، ولكننى حاولت ان اكون صارماً هنا حول المعانى. الدقة للكلمات المتعلقة ببعضها مثل object و class، أو declaration و definition وهذه الفروق الدقيقة وغيرها تهتم مؤلفى ال component الى حد كبير عن مستخدميها:

• **declaration:** هو عبارة لاسم عنصر، نوع، وال parameters الخاص به.

• **definition:** هو انشاء أو تنفيذ عنصر موضح.

• **class:** هى تعريف نوع جديد يضم بيانات و code للعمل على هذه البيانات. تمثل ال class نوع بيانات بسيط مثل نوع ال Object Pascal Integer. تبدأ اسماء ال Class عادة ب T (TButton، مثلاً).

● **Instance** : التعريف في الذاكرة لمتغير class قد يعرف البرنامج أى عدد يحتاجه من الinstances لـ class ما . وتشبه الinstances متغيراً لنوع بيانات بسيط مثل الInteger . يمكنك تعريف أى عدد يحتاجه برنامجك من متغيرات الInteger ، ولكن هناك نوع بيانات Integer واحد فقط . بنفس هذا المعنى ، يمكنك تعريف أى عدد يحتاجه برنامجك من الclass instances . (ولكن بعض الclasses النادرة قد تفيدك بإنشاء instances واحدة . على سبيل المثال ، يمكن ان يكون هناك TApplication instances واحدة فى أى تطبيق) .

● **Component** : هو class مشتقة بطريقة مباشرة أو غير مباشرة من الTComponent ذات مسارات وصول خاصة لبياناتها ، وخصائصها المنشورة ، ومميزاتها الأخرى التى تلتزم بمتطلبات Delphi للcomponents . إنك تقوم بتعريف وتنفيذ الcomponents فى الObject Pascal units .

● **Object** : تماماً مثل الinstance . فى النسخ السابقة من الBorland Pascal ، كان الobject هو والآن الclass فى Delphi ، مما يؤدى الى كثير من عدم الفهم فى الكتب والأدلة التى تنشر فى ذلك الوقت . فى هذا الكتاب ، لقد استخدمت كلمة object لتعنى بشكل محدد instance من class أو نوع آخر . لذا فمن الصواب ان يطلق على متغير الInteger انه object .

● **Access specifie** : هو واحدة من الكلمات الأساسية-private ، published ، public ، protected التى تقسم تعريف الclass الى قطاعات . ان عبارات البرنامج لها درجات متفاوتة من حقوق الوصول الى التعريف ، وهذا يعتمد على محددى الوصول التابعين لهم . ان التعريف الprivate تكون للاستخدام فقط فى الunit التى تعرف الclass ؛ وتكون التعريفات الprotected ايضاً متاحة للـ classes المشتقة ؛ اما التعريفات الpublic فمتاحة لكل المستخدمين ؛ والتعريفات الpublished هى خصائص تعرف قواعد الوصول الى بيانات class ، وبالنسبة للـ components المثبتة فى لوحة الVCI ، تظهر فى نافذة الObject Inspector .

● **Field** : هو متغير معرف فى class .

• **Method**؛ هو procedure أو function معرف في class .

• **Method implementation**؛ هو هيكل ال method ، بما في ذلك أى متغيرات أو عبارات تؤدي اعمال ال method .

• **Property**؛ هو حقل published يحدد بشكل نموذجي methods لقراءة وكتابة قيمة الحقل " وكذلك ، هو published event handler مثل ال OnClick .

• **Event**؛ هو procedure ل pointer متوفر بواسطة المستخدم ل objects معينة . قارن هذا بال method الذى يؤدي العمليات المتطابقة لكل ال object لل method class . ان خاصية ال event تمكن ال object ان يرسل اعمال الى object آخر ، والذى يمكن ان يؤدي عمليات متنوعة لنفس نوع ال event . على سبيل المثال ، يمكن ان يستجيب اثنين من ال TButton يعمل بنفس الطريقة مع كل ال Button objects .

• **Unit**؛ هى Pascal module تم تخزينها وعمل compiled منفصلة عن ال modules الاخرى . قد تحتوى units على component واحد أو أكثر . ولكن ، يمكنك تثبيت units كاملة فقط ، متضمنة كل ال components ، لوحة . ال component . لتثبيت components منفردة ، فيجب ان يكونوا فى units منفصلة .

• **Interface part**؛ هو التوضيحات التى يمكن التشارك فيها بشكل عام فى units ما ؛ والتى تسبقها الكلمة الاساسية interface .

• **Implementation part**؛ هى التعريفات والتعليقات الخاصة التى لا يمكن التشارك فيها ، وتكون مسبقة بالكلمة الاساسية Implementation .

• **Initialization part**؛ هى عبارات تمر تنفيذها عندما تم تحميل ال module فى الذاكرة ؛ وتكون مسبقة بالكلمة الاساسية initialization ومتتية بالكلمة الاساسية end ونقطة .

## الباب العشرون : إنشاء custom Components

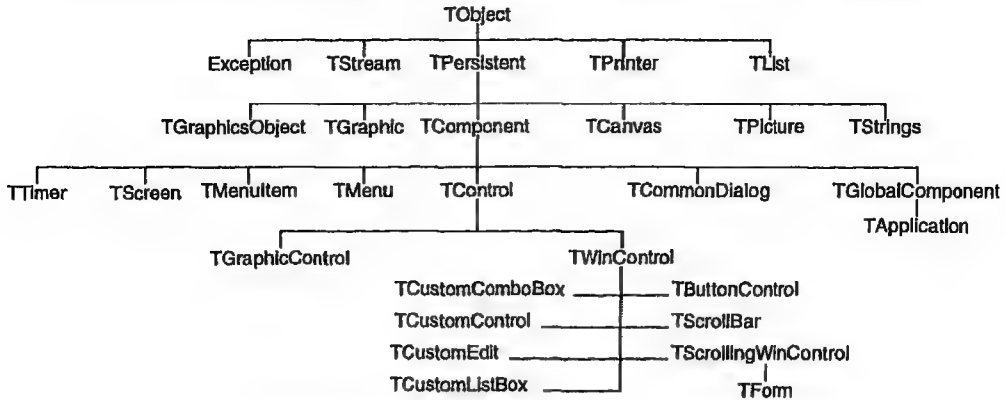
● **Visual component**، هو component له تمثيل بصرى فى وقت التشغيل . بعض ال components مثل ال TClipboard class تكون غير بصرية، ولكنها لا تزال components فى كل النواحي الاخرى . قد تظهر ال components البصرية أو لا تظهر على ال palette . على سبيل المثال، يعتبر ال TForm components بصرياً، ولكنه غير متاح على ال palette .

**ملحوظة:** من المفترض انك تدرك ماسأقوله الآن، ولكن الى الذين قفزوا الى هذا الباب، لقد اشرت فى هذا الكتاب الى ال components الموجودة على ال component palette باسماء نصها- مثل Button، ListBox وغيرها . ان classes هذه ال components تسمى TButton و TListBox . عندما يكون الاسم مسبقاً بحرف T، فإنه يشير الى تعريف لل class؛ وبدون ال T، فإنه يشير الى ال components كما يتم استخدامه فى مصمم Delphi .

### : Component class hierarchy

ان اغلب ال components الجديدة تستند الى classes ' components موجودة بالفعل . ويرث ال component المشتق كل الخصائص وال methods وال events من ال component السالف له . وللتمكن من إنشاء ال components، فإنك تحتاج ان تتعرف على مكتبة Delphi component و methods التى توفرها لل components يوضح الشكل (٢٠-١) الخاص بال hierarchy classes الذى يشكل ال components السابقة . استخدم هذا ال object لاستكشاف الخصائص وال methods وال events التى ترثها ال class المشتقة . حاول الا تعيد اختراع امكانيات يمكن ان ترثها من component class اخرى . وبالطبع ان ال hierarchy الموضح هنا ليس كاملاً . إنك تجد شكل الشجرة الكامل مع Delphi .

لكى تصبح مؤلف components خبير، فإنك تحتاج الى ال source files لمكتبة وقت التشغيل الخاصة بـ Delphi (ولكنك لا تحتاجهم لهذا الباب) . ان ال source files تشمل كل البرمجة الخاصة بمكتبة component البصرى، ويمكنك تعلم تقنيات وحيل عديدة بتصفح الملفات .



شكل (٢٠-١)، Delphi component class hierarchy

## معلومات حول الـ Packages:

قبل تطوير component جديد، إنك تحتاج ان تتعرف على packages .  
والـ package يحتوى على compiled units ، والتي توفر الاعمال الداخلية للـ  
components مثل الـ Buttons والـ ListBoxes . وإنشاء package لهذه الـ  
units يتم مهمتين أساسيتين :

- إنه يمكن تطبيقات متعددة من التشارك فى الـ VCL و component code  
اخرى، بدلاً من ان يطلب من كل تطبيق ان يكون له نسخة منفصلة من هذه  
الـ code.

- إنه يوفر وسيلة لتثبيت components فى Delphi palette . مع الـ  
packages ، ليس من الضرورى ان تعيد الـ VCL باكملها، كما كان  
الحال فى النسخ السابقة من Delphi ، لتثبيت component فى الـ palette .  
بالرغم من ان هاتين الميزتين تستخدمان المستندات إلا انهما مختلفتين الى حد  
كبير ويستخدمان package formats مختلفة لذا السبب يوجد نوعين من الـ  
packages :

- **packages وقت التشغيل:** تستخدم لتوفير component code تستطيع  
تطبيقات Delphi المتعددة ان تتشارك فيها .

• **packages وقت التصميم:** تستخدم لتثبيت components في Delphi ، وتوفر رابطة بمستندات وقت التشغيل المرتبطة بها .

يوضح هذا الفصل المفاهيم العامة لكيفية إنشاء واستخدام packages وقت التشغيل و packages وقت التصميم . إنك بحاجة الى ان تفهم اساسيات ال packages حتى تنشئ components جديدة بغرض تثبيتها على ال Delphi palette .

**فكرة:** لتشغيل packages وقت التشغيل ، استخدم **Tip** ProjectOptions ، اختر ال Packages page tab ، وقت بتشغيل ال Build مع packages check box وقت التشغيل يمكنك عندئذ بناء تطبيقاتك بحيث تشارك في ال code في units تكون موضوعة في مراجع من قبل ال packages . وعند إبطال هذا الخيار ، يتم نسخ ال unit code مباشرة الى ملف ال code .exe

ان إنشاء ال packages . تعتبر أمراً بسيطاً . أولاً ، قد بإنشاء واختبار وإزالة اخطاء ال units التي تريد تثبيتها في packages . ان packages وقت التشغيل هي الاسهل من حيث الإنشاء لذا فمن الافضل ان تبدأ بها . قم بوضع units واحدة أو أكثر تريد تثبيتها في packages ، ثم اتبع هذه الخطوات :

١- ابدأ تطبيقاً جديداً يمكن ان يكون هذا التطبيق هو البرنامج الاختياري لل packages أو ال units الخاصة بك ، ولكن اذا كنت لا تحتاج واحداً ، يمكنك إغلاق وتجاهل ملفات مشروع و units التطبيق .

٢- اختر ..... FileNew واضغط ال New page tab . اضغط مرتين ايقونة ال Package لتبدأ ال Package Editor الخاص ب Delphi .

٣- ان ال New Package dialog يسأل عن اسم ملف والوصف . اضغط زر ال Browse لفتح dialog ملف معياري ، وانتقل الى الدليل حيث تريد تخزين ملفات المستند . قم بادخال اسم ملف في هذا ال dialog بدون إمتداد اسم ملف ، على سبيل المثال ، TestPkg . اضغط Enter ، ويدخل ال Delphi مساراً واسم ملف زائد الامتداد ب dpk.. في مربع اسم الملف . أدخل وصفاً اذا اردت - هذا يعتبر اختيارياً ، ولكنه يظهر في اماكن متنوعة ، فمن المستحسن ان تدخل واحداً .

٤- اضغط زر ال OK لل New Package dialog . هذا يؤدي الى إظهار نافذة ال Package Editor الخاصة بـ Delphi . لإضافة units الى ال Package ، اضغط زر ال Add ، وتصفح ملف ال pas source code . الخاص بال unit . لاحظ انك تضيف ملفات source code ، وليس ملفات compiled unit ، الى ال Package ، اضغط زر Ok واضغط Enter للعودة الى ال Package Editor . اضغط ال Add ، مرة اخرى لأى عدد من ال units تريد ان تضيفها الى ال package .

٥- عندما تنتهى من اضافة ال units ، اضغط ال Requires page tab الخاص بالمحرر . هذا يؤدي الى اظهار ال package وهى تحتوى على ال units التى تحتاجها و units . اخرى فى ال package الجديد . وكحد أدنى ، فإن ال package تحتاج الى ان تشير الى VCL40 والذى يحتوى على ال VCL units . لإضافة Package اخرى . اضغط زر ال Add وقم بالدخول فيه أو تصفح ملف package آخر .

٦- عندما تنتهى من إضافة ال units المحتواه وال package المطلوبة ، اضغط زر ال Options . اضغط ال Description page tab واختر واحداً من ال check boxes الذان يحملان labeled Usage options أو كلاهما . للحصول على هذا العرض ، قم بتشغيل ال Runtime package option فقط . لإنشاء وتثبيت components فى ال Delphi palette ، فإنك تقوم بتشغيل كلا الخيارين . قم بتحرير الخيارات الاخرى كما تدعو الحاجة لل units و component . اضغط OK لإغلاق ال Package Options dialog .

ان هذه الخطوات تتم تصميم ال package الجديد . (ان ادخال components فى ال package وال Delphi palette يتطلب بعض الخطوات الإضافية ، والتى ساوضحها حالاً) . إنك الآن مستعد لـ Compile واستخدام ال package الجديد الخاص بك . اضغط زر ال Compile لـ Compile ال package ، وكل ال units هذا يؤدي الى إنشاء ال bpl package . فى الدليل المخصص .

لاستعراض ال source code الخاصة لل package اضغط يميناً داخل نافذة ال Package Editor ، واختر أمر ال View Package Source . هذا يفتح ال

## الباب العشرون : إنشاء custom Components

package في نافذة محرر Delphi code . ويتجاهل السلسلة الطويلة من خيارات ال Compile ، والتي يتم إدخالها بصورة تلقائية في النص ، يعتبر ال package ببساطة عمل بسيط يبدو كهذا :

```
package TestPkg;  
{$R *.RES}  
{$ALIGN ON}  
{$ASSERTIONS ON}  
...  
{$DESCRIPTION 'Test runtime package'}  
{$DESIGNONLY}  
{$IMPLICITBUILD ON}  
  
requires  
    vcl40;  
  
contains  
    Clrform;  
  
end.
```

لقد استبدلت أغلب أوامر ال compilation بزر بيضاوي- سوف ترى المجموعة الكاملة عندما تستعرض ال source code للمستند الخاص بك .

تعتبر ال Packages ملفات compiled DLLs بشكل خاص . ولتمييزها عن ملفات ال dll . الأخرى ، فإن امتداد اسم ملف ال package هو .bpl . يحتوي ملف package source على ثلاثة عناصر أساسية :

- الكلمة الأساسية package واسم الملف متبوعاً بفصلة منقوطة .
- تعريف ال required الذي يحتوي ال packages الأخرى التي يطلبها هذا ال packages .
- تعريف ال contains الذي يحتوي ال units التي يجب ضمها في هذا ال package .



**ملحوظة:** بالرغم من التطبيق يكون مرتبطاً بـ packages وقت التشغيل، إلا أنه يجب على modules البرنامج أن تضيف الـ units التي تحتاجها إلى الـ uses. إن الـ Compiling مع packages وقت التشغيل يخبر التطبيق فقط أين يجد الـ component code. يجب أن تستمر الملفات الـ source للتطبيق في إدخال الـ units التي تطلبها بذكر أسماء الـ units في تعريف الـ uses.

إن الـ Components التي تريد تثبيتها على لوحة Delphi يجب أن تكون compiled باستخدام الـ packages. والخطوات المطلوبة لعمل هذا تشبه الخطوات الموضحة لـ packages وقت التشغيل، ولكنك عادة ماتختار مخرجات مستندات الـ Runtime packages والـ Design-time components لهذا يؤدي إلى إنشاء packages: واحداً للاستخدام في وقت التشغيل للتطبيقات الـ packaged components، وواحداً للاستخدام في وقت التصميم لتشغيل الـ component على الـ VCL palette.

قد يكون من قبيل الفائض عن الحاجة أن يكون هناك نوعين مختلفين من الـ packages، ولكن في الواقع، فإن packages وقت التصميم تشير إلى packages وقت التشغيل. إن packages وقت التصميم من مجرد غلاف- فإن الـ code الفعلية للـ unit تكون دائماً في packages وقت التشغيل هذا يعني أنه بإمكانك توزيع packages وقت التصميم للمبرمجين الذين يحتاجون إلى استخدام الـ components بإختيارها من VCL palette الخاصة بـ Delphi.

لفتح ملف package لإعادة الـ Compile، اختر امر الـ File|Open، واستعرض ملفات من نوع Delphi package source (\*.dpk).

**ملحوظة:** لاستخدام package وقت التصميم-والذي يجب ان تفعله لكل الـ components المثبة على الـ VCL-يجب ان يكون لديك package وقت التصميم المناسب وpackage وقت التشغيل المتعلق به.

يوضح الفصل التالي كيفية إنشاء component جديد، وادخاله في package، وتثبيت الـ VCL palette component الخاصة بـ Delphi الـ Samples.

## تطوير الـ Component:

فى الفصول التالية ، تقوم بإنشاء إثنين من الـ components التى توضح برمجة الـ component وعملية تثبيت component تام فى Delphi palette . كل هذه الخطوات يمكن الرجوع عنها .

اولاً ، تقوم بإنشاء component بسيط فقط لتوضيح بعض الخطوات المطلوبة . بعد ذلك ، تقوم بإنشاء component أكثر تعقيداً يوضح المزيد عن الخصائص و الـ methods والـ events ، زائد تقنيات الـ registration والـ debugging والـ installation .

### الخطوات الأولى:

تتطلب الـ components تقنيات برمجة مختلفة عنها فى التطبيقات . يمكنك برمجة components من لاشئ باستخدام محرر Delphi code ، ولكن لان الـ components التى سيتم تشبيها على الـ VCL palette تتطلب التثبيت فى الـ Package ، فمن المستحسن ان تستخدم الـ Package Editor الخاص بـ Delphi لإنشاء الـ component unit يجب ان تستخدم هذا المحرر على ايه حال لـ compile وتثبيت الـ component فى الـ palette ، لذا فيمكنك ايضاً استخدامه لإنشاء الـ unit .

إتبع هذه الخطوات لتقسيم الـ Button component القياسى component جديد يحمل اسم TDingButton . وهذا الـ component يقوم بإنشاء زر والذى ، عندما يضغط ، يدق جرس الحاسوب . هذا الاختيار يوضح إثنين من الجوانب الهامة فى بناء الـ component : وراثة إمكانات من component موجود بالفعل والذى تبنى عليه الـ component الجديد ، واستخدام الـ Package لتثبيت component على الـ VCL palette .

**على القرص المدمج:** ان الدليل Source\Ding على القرص المدمج يحتوى على كل الملفات الخاصة بهذا الفصل . اذا لم تكن تريد إنشاء الـ component الخاص بك باتباع الخطوات المذكورة هنا ، افتح ملف الـ Ding package ، واستخدم الـ Package Editor لـ compile وتثبيت الـ



component فى Delphi. ولكن، لكى تفهم العملية جيداً، فإننى اقترح عليك ان تنشئ دليلاً جديداً وتتبع هذه الخطوات لبناء الـ TDing class الخاصة بك.

١- قم بإنشاء دليل (لقد اعطيك الدليل الخاص بى اسم Ding لتحميل الـ component واختيار ملفات البرنامج. إبدأ تطبيقاً جديداً. إجعل اسم form MainForm. اختر File|Save all... واحفظ الـ Unit1 على انها Main.pas، والم شروع على انه TestDing.dpr فى الدليل الخاص بك. يكون المشروع بمثابة برنامج اختياري للـ component.

٢- اختر File|New... واضغط الـ New page tab. اضغط مرتين ايقونة الـ Package.

٣- باستخدام الـ New Package dialog، اضغط Browse وتحقق من ان الدليل من الخطوة رقم (١) مازال جارياً. إدخل DingPkg فى حقل اسم الملف، واضغط Open. هذا يؤدي الى ادخال المسار واسم الملف DingPkg.dpk فى الـ New Package dialog. واختيارياً، قم بادخال وصف مثل "Button with a bell،" واضغط OK للاستمرار.

٤- ان الـ Package Editor الآن فى حالة تشغيل. اضغط زر الـ Add، عندما يظهر الـ Add dialog، اضغط الـ New Component page tab. هذا يؤدي لفتح dialog آخر يمكنك ان تحدد فيه صفات الـ component الخاص بك. إدخل أو اختر من قوائم اللائحة المعلومات التالية:

- TButton : (Ancestor type).
- TDingButton : (Class name).
- Samples : Palette page (الافتراضى).
- C:\...\Ding\DingButton.pas : (Unit file name) (يتم ادخاله بصورة تلقائية).
- (Search path) : (يجب ان يحتوى على مسارات افتراضية زائد المسار الخاص بـ Ding).

## الباب العشرون : إنشاء custom Components

٥- اضغط OK لإنشاء غلاف الـ component. هذا يقفزك الى الخلف الى Package Editor. اضغط الـ Requires وحدد أى package مطلوبة- لهذا. فى هذا المثال، فإن vcl340 package الافتراضى هو كل ماتحتاجه. فى الـ components الخاصة بك، قد تريد ان تضيف package اضافية الى هذه القائمة.

٦- اضغط زر الـ Options وقم بتشغيل كلا من Design و Runtime check boxes. قد تغير فى الخيارات الأخرى، ولكن لهذا العرض، فإن القيم الافتراضية لأبأس بها اضغط OK للعودة الى الـ Package Editor.

٧- اضغط زر الـ Compile لإجراء عملية الـ compile لـ package والـ component unit الخاصة بك. هذا يؤكد ببساطة ان الـ format والـ unit. وفى الـ package الصحيحة لهما. عندما تكون مستعد لتثبيت الـ component التام، اضغط Compile مرة أخرى، ثم اضغط Install لادخال الـ component فى الـ palette. ولكن لا تفعل هذا الآن-اولاً، نريد ان نبرمج ما يقوم به الـ component الجديد.

توضح القائمة (٢٠-١) ملف الـ DingButton.pas التام. انتقل الى نافذة محرر الـ Delphi code، واختر الـ DingButton page tab. استخدم القائمة كمرشد لك للملئ غلاف الـ component الذى انشأته فى الخطوات السابقة.

```
القائمة (٢٠-١) : Ding\DingButton.pas
unit DingButton;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls,
  Forms, Dialogs, StdCtrls;

type
  TDingButton = class(TButton)
  private
    { Private declarations }
  protected
```

```
{ Protected declarations }
public
procedure Click; override;
{ Public declarations }
published
{ Published declarations }
end;
```

procedure Register;

implementation

```
procedure Register;
begin
    RegisterComponents('Samples', [TDingButton]);
end;
procedure TDing.Click;
begin
    MessageBeep(0);
    inherited Click;
end;
end.
```

يضيف البرنامج الثام سطور قليلة فقط الى قالب ال component . لتوفير صوت جرس عندما يضغط المستخدم زر ، فقد عرفت ال Click procedure فى ال TDing class مع امر (override):

procedure Click; override;

ان ال TButton class الذى يستند عليها ال TDing تعرف ال procedure Click مع العديد من الخصائص وال methods الاخرى . ان تعريف ال override يخبر ال compiler ان ال component الجديد يحل محل ال Click الموروث مع البرمجة الخاصة به . بالإضافة الى تعريف ال method ، يجب عليك ايضاً تنفيذ بكتابة هيكل ال procedure . فى هذا المثال ، فإن ال Click method المستبدل ينفذ امرين :

```
MessageBeep(0);  
inherited Click;
```

ان السطر الاول يصدر صوتاً باستدعاء الـ Windows API MessageBeep function . والسطر الثانى يستدعى الـ Click procedure المشتق لأداء اعمال الـ TButton ، والتي تشمل تنفيذ الـ OnClick اذا تم تعيين واحداً . عند ابطال method ، فإنك دائماً ما تستدعى الـ method المشتقة . هذه هي كيفية بنائك لإمكانات جديدة فى الـ component مع الاحتفاظ بما تقوم به الـ methods بالفعل . وليس مطلوباً منك ان تستدعى method مشتقاً ، ولكن اذا لم تفعل ، فقد يتخطى برنامجك شيئاً هاماً مدفوناً فى مكان ما فى الـ class الأم . للسلامة ، استدع دائماً الـ method المشتق فى الـ functions و procedures الـ الاحلال الا اذا كان لديك سبب قوى فى الـ لا تفعل .

لاحظ ان الـ DingButton.pas module تشمل الـ Register procedure والذي لا يعتبر عضواً فى الـ TDing class . يستدعى الـ Delphi هذا الـ procedure كجزء من عملية تثبيت الـ component palette الخاصة به اتبع خطوات تثبيت الـ componen الجديد (بعد ذلك ، سوف اشرح كيف تحذف الـ procedure ، لذا لا تتردد فى تجربة هذا) .

١- ارجع الى نافذة الـ Package Editor . اذا لم تستطيع العثور عليها ، اختر View/Window List... واختر Package Editor لإظهار هذه النافذة .

٢- اضغط زر الـ Compile . هذا يؤدي الى الـ Compile الـ package والـ unit المتعلقة به . تذكر ، يجب ان تقوم بـ Compile الـ package ان الـ Compile الـ unit فقط منفصلة لا يؤدي الى إنشاء سليم للـ package .

٣- اذا لم تتلقى ايه اخطاء فى الخطوة رقم (٢) ، اضغط Install لتثبيت الـ component على الـ VCL palette . يجب ان تتلقى رسالة مثل "Package D:\...\DingPkg.bpl has been installed. The following new component(s) have been registered: TDingButton." اضغط زر الـ OK لاجلاق نافذة الرسالة هذه .

اضغط ال Samples page tab على ال VCL palette الخاصة بـ Delphi .  
ان الزر الجديد موجود على page وجاهز للاستخدام لاختبار ال component  
الجديد، يمكنك انشاء تطبيق جديد، أو استخدم التطبيق الحالى الذى انشأته فى بداية  
لعملية باكملها . لاستخدام هذا التطبيق، افتح ال Project Manager اذا لزم الأمر  
واجعل ال Main form مرئية . أضف ال DingButton فى ال form . لاحظ ان  
Delphi يعطى هذا ال object اسم ال DingButton1 . لاحظ ايضاً اذا اوقفت المؤشر  
على ايقونة ال component ، فإن Delphi يعرض اسمه فى نافذة .

اضغط F9 لـ compile التطبيق . اضغط DingButton لتدق جرس  
الحاسوب الخاص بك . قد تريد ان تحفظ كل الملفات فى هذه المرحلة قبل الاستمرار .  
لحذف component ، استخدم Component\Configure Palette...  
اختر فئة على اليسار (Samples مثلاً) ، واختر ال component الذى تريد ازالته  
(وهو Ding فى هذه الحالة) ، ثم اضغط Delete .

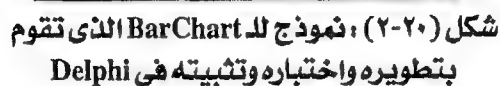
### الخصائص، والـ methods، والـ events :

ان كتابة components يتضمن برمجة ثلاثة عناصر اساسية تكون موجودة  
فى كل class فى كل component وهى : الخصائص والـ methods ، والـ  
events . فى هذا الفصل والفصول التالية، فإنك تقوم بتطوير، واختبار، وتثبيت  
components اكثر توسعاً والى تستخدم هذه العناصر الثلاثة . والنتيجة التامة ،  
وهو ال BarChart من مجموعة من نقاط البيانات تديرها قائمة ال string list .  
يوضح شكل (٢٠-٢) مثال للـ component المستخدم .

**على القرص المدمج:** اذا لم تكن تريد إنشاء ال component الخاص  
بك ، انسخ دليل ال Source\BarChart من على القرص المدمج ، اختر  
Component\Install Packages ، اضغط Add ، وتصفح دليل ال  
BarChart على محرك القرص الصلب الخاص بك . اختر ال BarPkg.bpl ،  
واضغط OK مرتين . ولكن، إننى اشجعك على اتباع التعليمات الموضحة هنا  
خطوة بخطوة ، والى توضح بطريقة افضل كيفية إنشاء وتثبيت ال components  
الخاصة بك . (ولكن مازال بإمكانك نسخ نص البرنامج من على القرص المدمج  
لتوفير الوقت) .



=====



١- قم بإنشاء دليل BarChart جديد. ابدأ تطبيقاً جديداً، اجعل اسم ال form MainForm، واحفظ جميع الملفات واحفظ ال Unit1.pas على انها Main.pas، واحفظ المشروع على انه Bartest.dpr. كما كان الحال مع زر ال Ding، يكون ال Bartest بمثابة برنامج اختباري للـ BarChart.

٣- إنك ترى الآن ال Package Editor. اضغط زر ال Add، ثم اختر ال New Component page tab. حدد الحقول الموجودة في صفحة ال dialog هذه بمايلي:

\* Samples : (Palette page) (الافتراضي).



\* (Unit file name) : C:\...\BarChart\BarChart.pas (تأكد مرتين من هذا المدخل - فقد يكون محدداً بدليل ال Ding من الفصل السابق . اذا كان كذلك ، قم بتغييره الى BarChart).

\* (Search path) : (يجب ان يحتوى على مسارات افتراضية زائد المسار الخاص بالBarChart؛ قم بحذف المدخل الخاص بال Ding اذا كان ضرورياً).

٤- اضغط OK لإنشاء غلاف ال component الجديد، وارجع الى ال Package Editor . اضغط ال Requires page tab وحدد أى Package يطلبها هذا ال Package . كما كان الحال مع ال Ding ، ان كل المطلوب هو ال vcl40 package الافتراضى . فى ال component ، قد تحتاج ان تضيف package اضافية الى هذه القائمة .

٥- اضغط زر ال Options وقم بتشغيل كلاً من ال Runtime check boxes و ال Design . ان القيم الافتراضية للخيار الآخر لا بأس بها . اضغط OK للعودة الى ال Package Editor .

٦- اضغط زر ال Compile لـ Compile ال package و ال component الخاصة به . هذا يضمن ان ال package و ال unit فى ال formats الصحيحة لهما . عندما تكون مستعداً لتثبيت ال component اضغط Compile مرة أخيرة ، ثم اضغط Install لإدخال ال component فى اللوحة . ولكن ، لا تفعل هذا الآن - كما كان الحال مع زر ال Ding ، فنحن نحتاج أولاً ان نبرمج ما يقوم به ال component الجديد .

٧- انتقل الى ال Delphi code editor ، واختر ال BarChart page tab . انسخ البرمجة من على القرص المدمج فى ملف BarChart.pas الموجود فى دليل ال Source\BarChart . ان باقى هذا الباب يشرح عبارات هذه القائمة - اما الآن ، استخدم النص كله ليحل محل ال BarChart.pas unit الفارغة فى ال code editor .

ان الخطوات السابقة تتم البرمجة للـ BarChart component . يمكنك الآن compile ال component وتثبيت ال package الخاص به على ال VCL palette . بعد ذلك ، يمكنك استخدام ال component كما تفعل مع ال

## الباب العشرون : إنشاء custom Components

components الأخرى . لا كمال ال component المخصص واختباره، إتبع هذه الخطوات :

١- ارجع الى نافذة ال Package Editor . اذا لم تستطيع العثور عليها، اختر View|Window List... واختر Package Editor لإظهار هذه النافذة .

٢- اضغط زر ال Compile . هذا يؤدي الى Compile ال package وال unit المتعلقة به. تذكر، يجب ان تقوم بـ Compile ال package ان Compile ال unit فقط منفصلة لا يؤدي الى إنشاء سليم لل package .

٣- اضغط Install لتثبيت ال component الجديد في ال VCL palette . يجب ان تتلقى رسالة مثل "Package C:\...\BarPkg.bpl has been installed. The following new component(s) have been registered: TBarChart." اضغط زر ال OK لإغلاق نافذة الرسالة هذه .

٤- قم بتجربة component الجديد . اضغط ال Main page tab في نافذة ال code editor، واضغط F12 لإظهار form هذا البرنامج الاختباري . اضغط ال Samples VCL، واختر ال BarChart . اضغط في ال form لإضافة ال BarChart .

٥- مع اختيار ال BarChart1، اضغط الزر البيضاوي الواقع بعد خاصية ال Data لل object . هذا يؤدي الى اظهار محرر قائمة ال string الخاص بـ Delphi . أدخل قيم floating-point من صفر الى مئة، مدخل واحد لكل سطر (٩، ٣٢، ٣، ١٢، ٥، ٧٥، ٧، ٩٨، الى آخره) .

٦- اضغط زر ال OK الخاص بمحرر القائمة للعودة الى ال form واستعرض ال chart المعروض بواسطة ال BarChart . يوضح شكل (٢-٢) ال BarChart الذي تراه على الشاشة (لا يجب عليك تشغيل أو حفظ البرنامج الاختباري، ولكن تستطيع اذا اردت) .

### ال TBarChart component class :

توضح القائمة (٢-٢٠) قطاع واجهة التطبيق لل unit BarChart.pas، بما في ذلك تعريف ال TBarChart class . كما ترى، فإن ال Barchart unit

تستخدم العديد من Delphi unit modules الاخرى، ويمكنك اضافة اسماء unit اخرى الى تعريف الـ uses. ولأنك لا تقوم ببرمجة form، فعند برمجة component، لا يستطيع Delphi ان يضيف بصورة تلقائية اسماء الـ units الى الـ uses كما يفعل مع التطبيق. اذا قمت باستدعاء procedure أو function، أو اذا حاولت ان تنشئ class لـ object، وتلقيت خطأ Unknown identifier، انظر نوع identifier في online help الخاصة بـ Delphi واضف unit الى uses في اعلى الـ unit module. يمكنك ايضاً اضافة تعريف uses خاص للـ unit module.

**على القرص المدمج:** على خلاف اغلب قوائم المشروعات الموجودة في هذا الكتاب، قائمة الـ BarChart مذكورة في هذا الباب في اجزاء. سوف اوضح كل جزء عندما تمر به. وبالقسط، توجد قائمة الـ BarChart.pas باكملها على القرص المدمج في دليل الـ Source\BarChart.



#### القائمة (٢٠-٢٠): Barchart\BarChart.pas interface:

unit Barchart;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics,  
Controls,  
Forms, Dialogs;

type

TBarChart = class(TGraphicControl)

private

FPen: TPen;

FBrush: TBrush;

FData: TStrings;

FLabels: Boolean;

XBase, YBase: Integer;

## custom Components : إنشاء الباب العشرون

```

XIncrement, YIncrement: Integer;
  procedure SetPen(Value: TPen);
  procedure SetBrush(Value: TBrush);
  procedure SetData(Value: TStrings);
  procedure SetLabels(Value: Boolean);
  function YData(N: Integer): Integer;
protected
  procedure Paint; override;
  public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
  published
  procedure StyleChanged(Sender: TObject);
    property Pen: TPen read FPen write SetPen;
    property Brush: TBrush read FBrush write
SetBrush;
    property Data: TStrings read FData write SetData;
    property Labels: Boolean read FLabels write
SetLabels;
    property DragCursor;
    property DragMode;
    property Enabled;
    property ParentShowHint;
    property ShowHint;
    property Visible;
    property OnDragDrop;
    property OnDragOver;
    property OnEndDrag;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
  end;

procedure Register;
```

ان ال TBarChart (ال class المشتقة) ترث خصائص و events و methods من ال TGraphicControl (ال class الأم) كما ترى من الشكل (٢٠-١)، ان ال class الام مشتقة من اربع classes اخرى: ال TControl ، ال TComponent ، ال TPersistent ، وال TObject . ولذلك فـإن ال TBarChart ترث كل خصائص و methods و events تلك ال classes . ولان العديد من ال components تكون مشتقة ايضاً من نفس هذه ال classes ، فإن كل ال methods وال events المشتقة تكون معلقة فى مكان ما فى تطبيقك . فإن وراثه هذه الاشياء تعنى ان ال class الجديدة يمكن ان تتوصل إليها .

تبدأ ال TBarChart class بعدة تعريفات خاصة . فإن كل الحقول ال methods التى تعقب الكلمة الاساسية private هى مخصصة للاستخدام من جانب ال unit التى تعرف ال class . ولا تشير أى unit اخرى فى أى مكان-أو component أو تطبيق-الى أى من التعريفات الخاصة للـ class . ويعد هذا مثالا على إخفاء البيانات ، وهو الأمر الذى يدرك المبرمجين الخبراء انه واحداً من اسس التطوير الناجح للبرمجيات . وتحدث (Bugs) نتيجة استخدام البيانات بلا تمييز ، وإخفاء عناصر هامة (مثل قائمة ال FData string الخاصة بال BarChart) يضمن أنه اذا ظهر أى خطأ بهذا الحقل ، فيجب ان تكون المشكلة فى هذه ال module ، لان هذه ال module وحدها تستطيع ان تشير الى ال FData . ان اخفاء البيانات ايضاً يجعل المطور حراً فى ان يقوم بتعديل ال class code دون قلق أن تؤثر هذه التغييرات على modules أخرى .

ولكن ، تحتاج ال TBarChart class أن تجعل قائمة ال FData string متاحة بحيث يستطيع مستخدمى ال component أن يوفرُوا بيانات جديدة لـ bar chart objects . وتقوم ال class بهذا العمل من خلال نشر الحقل ، والذى يحدد ال methods الوصول التى تحكم كيفية قراءة المستخدمين للبيانات الهامة وكتابتها-ولكن ، سأوضح المزيد عن هذا فيما بعد . أولاً ، أريد أن أشرح الأجزاء الخاصة الأخرى للـ TBarChart .

إن حقول البيانات الخاصة التى يجب أن يتم نشرها تكون عادة مسبقة بالحرف F . ورغم أن هذا التقليد ليس مطلوباً ، إلا أنه يساعدك على إقترفاء أثر

## الباب العشرون : إنشاء custom Components

الاسماء الـ publish الخاصة . على سبيل المثال ، إن اسم الخاصية الـ publish الـ FData هي Data . فى برمجة الـ component ، يجب أن تكون قادراً على التمييز بين الاسماء الـ publish والخاصة للحقل .

تعرف الـ TBarChart class حقول الـ FPen ، FBrush ، FData ، و FLabels . قم بتجربة هذه الخصائص فى البرنامج . على سبيل المثال ، اضغط مرتين خاصية الـ Brush فى نافذة Object Inspector ، وقم بتغيير القيمة الفرعية Color لها . وكذلك ، قم بربط خاصية الـ Labels من True و False لمسح labels نقطة البيانات فوق الـ bars .

تعرف أيضاً الـ TBarChart class بعض المتغيرات المتنوعة فقط للملائمة . وبتعريف هذه المتغيرات بعد الوصول إلى القطاع الـ private ، يمكن أن تكون واثقاً من أن أى module أخرى لن تستطيع التأثير على الـ TBarChart objects إلا من خلال الـ methods التى توفرها الـ class بشكل خاص .

بالإضافة الى الحقول ، أيضاً القطاع الـ private بالـ class تعرف خمسة methods : أربعة procedures و function واحدة . تقوم الـ Set procedures بتعيين قيم الى الحقول الخاصة بها . فالـ SetPen يعين قيمة للـ FPen ، والـ SetBrush يعين قيمة للـ FBrush ، الـ publish باستخدام هذه الـ methods لكتابة قيم الحقول المرتبطة بها . وتقوم الـ YData function بإدخال إحداثية عدد صحيح لنقطة بيانات مفهرسة بـ N .

ومرة أخرى نقول أن كل هذه الـ methods تكون الـ private للـ class . هذا يعنى أن العبارات الموجودة فى هذه الـ unit فقط تستطيع أن تستدعيها . ولا يمكن لأى modules أخرى أن تستدعي الـ SetPen أو الـ SetLabels . هذا يقيد استخدام الـ component ويساعد على منع الأخطاء . ولكن ، لأن الخصائص تكون الـ public ، فإن التعيينات إليها باستخدام الـ Object Inspector تؤدي الى استدعاء الـ SetPen والـ SetLabels .

**Tip** فكرة: فى القطاعات الـ public ، والـ protected والـ private للـ class ، يجب أن تسبق تعريفات الحقول تعريفات الـ method والخاصية .

وبعد القطاع private للـ TBarChart class، تقوم الـ class بتعريف procedure استبدالاً، وهو Paint، فى القطاع الـ protected. وترث الـ TBarChart class الـ Paint procedure من الـ class السالفة لها، ولكى تستبدل هذا الـ method، فيجب ان تنهى التعريف (بالـ override). ان الـ methods والحقول المقررة فى القطاع الـ protected تستطيع الـ unit ان تستخدمها، وكذلك أى classes مشتقة من الـ TBarChart.

ولكن لا يستطيع مستخدمى الـ component استدعاء الـ Paint protected مباشرة. وهذا يوفر وسيلة أمنية لأن الـ Paint يتم استدعاءه استجابة لتلقى النافذة رسالة wm\_Paint. وتحل الـ class الجديدة محل الـ Paint بحيث تستطيع رسم bar chart، ولكن سوف يكون من قبيل الخطأ أن تستدعى التطبيقات هذا الـ procedure مباشرة يمكنك تعريف حقول و methods وخصائص فى القطاع الـ protected للـ class.

وتعرف الـ TBarChart class أثنان من الـ method آخرين: الـ Constructor (create) والـ destructor (destroy). وكلاهما مشتقان، وكلاهما يحتاج الى الـ override. وإنهما معرفان فى القطاع الـ public للـ class بحيث يستطيع المستخدمون استدعاءهما. اذا لم تضع الـ Create فى القطاع الـ public، فلن يكون أمام البرامج سبيلاً لإنشاء الـ TBarChart. يمكنك تعريف حقول و methods وخصائص فى القطاع الـ public. وأى عبارة فى أى مكان من التطبيق الذى يستخدم الـ TBarChart يمكنها استدعاء methods عامة، وقراءة وكتابة أى حقول بيانات عامة.

وأخيراً فى الـ TBarChart يوجد قطاع طويل مسبق بتعريف الـ published. فى هذا القطاع توجد أسماء الخصائص التى تريد من Delphi أن يوفرها فى نافذة الـ Object Inspector للـ objects الخاص بالـ class. وأى methods تعتبر public - مثل الـ StyleChanged، ولكنك غالباً ما تنشر فقط أسماء events وحقول البيانات.

وتعتبر بعض خصائص الـ TBarChart published جديدة، وبعضها موروثاً من class سالفة. على سبيل المثال، ان خاصية الـ Visible متاحة فى كل الـ

## الباب العشرون : إنشاء custom Components

components البصرية اذا لم تكن تريد ان يقوم مستخدمى ال components بتغيير قيمة ال Visible ، فلا تضعها فى قطاع ال published . ان تحديد أى الخصائص يجب أن تكون ال published امر يرجع إليك ، ولكنك غالباً أغلب الخصائص الموروثة فى ال ال published ، مثل ال Enabled وال Visible التى يتوقع المستخدم ان يجدها فى ال . Object Inspector .

ان الخصائص الجديدة على ال TBarChart تستخدم نوع مختلف من تعريف الخاصية والذى يحدد مسار الوصول الى البيانات . على سبيل المثال ، دقق النظر فى التعريف ال published لخاصية ال Pen :

property Pen: TPen read FPen write SetPen;

فهناك نقطتان بعد اسم الخاصية Pen ، بعدهما يوجد تعبير يخبر عن نوع بيانات الخاصية (TPen) ، methods الوصول لقراءتها وكتابتها . وتستطيع عبارات ال . Object Inspector مباشرة تعيين قيم TPen ، بحيث يكون method الوصول للقراءة هو ببساطة اسم حقل بيانات ال FPen . ولكن method الكتابة write يعتبر method خاصاً فى ال SetPen class . هذا يعنى ان كل المواصفات لخاصية ال Pen تستدعى ال SetPen بالفعل .

وتستخدم الخصائص الأخرى برمجة مشابهة . بوضعها فى قطاع ال published وتوفير methods الوصول للقراءة والكتابة ، فإنك تتحكم بحذر فى استخدام بيانات ال component . ان ال TBarChart ما هو الا مثال بسيط نسبياً ، ولكن فى الحالات الأكثر تعقيداً ، فإن هذا التحكم لا يقدر بثمن على سبيل المثال ، يمكنك برمجة ال Set method لمنع المستخدمين من تعيين قيم غير مسموح بها أو خارجة عن نطاق للمتغيرات .

وهناك طرق أخرى متاحة للخصائص ال publish اذا قمت بتوفير method ل (read) فقط ، فتكون الخاصية read-only . يمكنك أيضاً إنشاء خاصية write-only لتقييد الحقول الهامة مثل كلمة المرور ، ولكن هذه الاستخدامات تكون نادرة . ويمكن لعبارات الخاصية ان يكون لها اوامر إضافية للقيم الافتراضية وتحدد ما اذا كان يجب تخزين الخاصية فى ملف form . وهذه التقنيات تعتبر خارج نطاق مقدمة هذا الباب ، لكن يمكنك القراءة عنها فى ال online help Delphi أو فى ال Developer Guide .



والسطر الاخير فى واجهة تطبيق ال Barchart unit يعرف ال procedure  
والذى يجب ان يكون لدى ال component units للتثبيت على . ال palette ويقوم  
ال Component Expert بادخال هذا التعريف بدلاً منك ، ولكن اذا كنت  
component من لا شىء ، يجب ان تذكر ان تدخل هذا السطر :  
procedure Register;

يقوم Delphi باستدعاء ال Register عندما تقوم بتثبيت ال component  
على ال palette وال unit التى تفتقد ال Register procedure لا يمكن تثبيتها . يتم  
استدعاء ال Register فقط عندما يتم ربط ال component فى ال palette . لا  
يجب ان يقوم البرنامج ابداً باستدعاء ال Register ، والذى يزيله ال Delphi  
linker من التطبيقات التى تستخدم ال component module .

### : Component class registration

توضح القائمة (٢٠-٣) تنفيذ ال Register الخاص بال Barchart unit .  
يقوم ال procedure باستدعاء ال RegisterComponents ، والذى يتطلب  
arguments : اسم ال component page ، ومجموعة من ال component  
classes فى ال unit ضع اقواساً مربعة حول واحداً أو أكثر من اسماء ال  
classes لإنشاء ال set .

القائمة (٢٠-٢) ، تنفيذ ال Register procedure الخاص بال Barchart Unit

```
{ Delphi calls this to install component onto the palette }
procedure Register;
begin
  RegisterComponents('Samples', [TBarChart]);
end;
```

اذا كانت ال unit لها component classes متعددة ، قم بإدخالها فى  
مجموعة اسم ال component ، تفصلها الفاصلات . على سبيل المثال ، فى ال unit  
ذات ثلاثة classes ، يمكنك استدعاء ال RegisterComponents كهذا :

```
RegisterComponents('Samples', [ThisCtrl, TherCtrl,  
TYourCtrl]);
```

## الباب العشرون : إنشاء custom Components

أو، يمكنك استدعاء الـ RegisterComponents عدة مرات من الناحية العملية تعتبر هذه العبارات تماماً مثل العبارة السابقة، ولكن اسهل في التحرير في حالة ما اذا اضطررت الى تغيير عدد الـ components تسجلها الـ unit. وكذلك استخدم هذه الـ form لتثبيت components على pages مختلفة (يجب عليك ان تضع اقواساً حول اسماء الـ components):

```
RegisterComponents('Dialogs', [TThisCtrl]);  
RegisterComponents('Additional', [THerCtrl]);  
RegisterComponents('Samples', [TThisCtrl]);
```

لا يجب عليك تسجيل كل class في unit، فقط تلك الـ classes في الـ unit التي تريد تثبيتها على components. قد توفر الـ unit ايضاً procedures، functions، ومتغيرات، و classes، و components غير بصرية لا تحتاج ان يتم اختيارها من على اللوحة في وقت التصميم.

### :Component initialization

ان الـ Create constructor الخاص بال component يبدأ لـ class object. لكل component يجب أن يكون Create constructor. اذا لم يكن لـ component الخاص بك شيئاً للبدء، فإنك لا تحتاج الى إبطال الـ Create constructor الذي ترثه من الـ class السالفة، ولكن نادراً ما يكون هناك component لا يجب عليه أداء بعض العمليات عند الإنشاء.

وبنفس الطريقة، فإن كل الـ component تقريباً تحتاج الى الـ Destroy لتحرير أى resources للـ component objects. على سبيل المثال، اذا قمت بإنشاء object أو خصصت بعض الذاكرة في الـ component constructor، فيجب أن تضيف deallocation في الـ destructor ومن المؤكد أن يؤدي هذا إلى فقد نهائي للذاكرة (وهو ما يسمى بـ memory leak). توضح القائمة (٤-٢٠) constructor و destructor الـ TBarChart.

القائمة (٤-٢٠): constructor و destructor الـ TBarChart

```
{ Create component instance at runtime AND design time }  
constructor TBarChart.Create(AOwner: TComponent);
```

```

begin
  inherited Create(AOwner);
    Width := 65;
    Height := 65;
    FPen := TPen.Create;
    FPen.OnChange := StyleChanged;
    FBrush := TBrush.Create;
    FBrush.OnChange := StyleChanged;
    FData := TStringList.Create;
    FLabels := True;
end;

{ Destroy component instance at runtime AND design time
}
destructor TBarChart.Destroy;
begin
  FPen.Free;
  FBrush.Free;
  FData.Free;
  inherited Destroy;
end;

```

يجب أن يستدعى الـ Create دائماً الـ constructor الخاص به، عادة مثل الخطوة الأولى. وبخلاف methods الموروثة، يجب أن تقوم باستدعاء الـ Create constructor الموروث لا تغفل هذه الخطوة أبداً، أولاً يتم بدء object بعد استدعاء الـ Create الموروث، يعين الـ constructor قيم لخصائص الـ Width والـ Height، الموروثة من الـ class السالفة. وهذه القيم تحدد المظهر الأول للـ object عندما يضافه المستخدمون إلى الـ form.

بعد ذلك، يقوم الـ constructor بإنشاء حقول الـ FPen في الـ FBrush والـ FData وجميعها Delphi classes objects ويعتبر هذا مثلاً جيداً لكيفية إستطاعة الـ component unit أن تتوصل إلى التعريفات الخاصة لـ class ما. بالإضافة إلى إنشاء الحقول، يعين البرنامج procedure وهو الـ StyleChanged،

## الباب العشرون : إنشاء custom Components

للد OnChange event الخاص بال FPen وال FBrush. واستخدام ال Object Inspector لتعيين قيم لتلك الحقول يؤدي الى استدعاء ال StyleChanged ، والذي يجعل النافذة غير صالحة (جرب هذا مع برنامج). ولهذا السبب ، فعندما يقوم المستخدمون بتغيير لون خاصية ال Brush ، مثلاً ، يكون التأثير فوري .

وكخطوة اخيرة يحدد ال component قيمة افتراضية لحقل ال Boolean ، FLabels . ويمكن لعبارة الخاصة بال published في تعريف ال class ان تحدد نفس هذه القيمة لمنع كتابة هذا الحقل الى ملف ال form اذا كانت قيمة الخاصة ملائمة . ولان constructor بين هذه القيمة ، فلا يوجد سبب لهدر مساحة في كتابة القيمة الافتراضية الى ملف ال form . للقيام بهذا ، قم بتغيير تعريف الخاصية الى مايلي (هذا سطر واحد بالرغم من انه موضح هنا على سطرين):

```
property Labels: Boolean read FLabels write SetLabels
default True;
```

كما ذكرت ، يجب ان يقوم ال destructor الخاص بال class بتحرير أى resources . ان ال destructor الخاص بال TBarChart ، وهو Destroy ، يقوم بهذا العمل باستدعاء Free لحقول ال FPen ، وال FBrush ، وال FData التي انشاها ال constructor وكخطوة اخيرة ، يستدعى ال destructor خاصية ال Destroy . يجب على ال Components ان تفعل هذا حتى يتم تحرير أى مواصفات لـ class سالفة . لا تغفل هذه الخطوة ابدا .

وأي exception تحدث في ال constructor تترك ال object وقد تم بدئه جزئياً . يجب ان تبرمج ال destructor على ان يعالج هذا الاحتمال - على سبيل المثال ، بالتأكد من ان ال pointers قد تم بدؤها تحرير الذاكرة المخصصة لها .

ولا يتم فقط استدعاء ال component class الخاص بال component في وقت التشغيل ، ولكن ايضاً عندما يقوم المستخدمون بإضافة ال component object على form . وبنفس الطريقة ، اذا قام المستخدم بحذف ال object أو إغلاق ال form أو المشروع ، يستدعى البرنامج ال component destructor . ويمكن استدعاء . methods اخرى ايضاً - على سبيل المثال ، لعمل مواصفات للخصائص . ان ال Component objects تكون نشطة في وقت التشغيل واثناء تطوير التطبيق .

## :Visual component painting

لإعطاءها مظاهر رائعة على مسرح Delphi، يجب ان تتمكن ال components البصرية من ال Paint method الموروثة. ان ال method يستدعى ال Canvas functions، ويحدد ال pen وال brush color، ويقوم بكل ما هو ضرورى لرسم ال component object.

ان Delphi يهتم بتوفير ال methods، ومميزات ال clicking والسحب، وإعادة تحديد الحجم، والمميزات الأخرى لوقت التصميم. ان كل ما تحتاج إليه هو ان تعرف كيف تريد ان يبدو ال component object الخاص بك. توضح القائمة (٢٠-٥) تنفيذ ال Paint method TBarChart مع دعم الثوابت وال YData function، والتي تحول نقاط البيانات الى قيم أحداثية ال لرسم chart bars.

ويقوم ال Paint باستدعاء ال Windows GDI functions من خلال خدمات ال Canvas object الموروثة من ال TGraphicControl. لرسم نافذة ال BarChart- وهو اول ما يراه المستخدم عند إضافة ال BarChart على form- يقوم ال Paint بتعيين اللون ال Pen وال Brush الخاصة بال Canvas باستخدام حقول البيانات ال FPen و FBrush، والتي يستطيع المستخدم برمجتها فى ال Object Inspector. بعد اعداد القليل من المتغيرات، يقوم ال Paint باستدعاء ال Rectangle، وكالمسحر، يظهر مربع ابيض فى ال form. هذا هو كل ما تحتاج ان تفعله لتعطى ال components المظهر الخاص بها فى وقت التصميم.

**القائمة (٢٠-٥)، تنفيذ ال Paint method الخاص بال TBarChart**

```
const
{ Fixed constants }
numClrs = 16;           { Number of colors in colorArray }
spaceAtBottom = 10;    { Reserved pixels below chart }
spaceAtLeft = 20;      { Reserved pixels at left of chart }
spaceAtTop = 40;        { Reserved pixels above chart }
spaceAtRight = 20;     { Reserved pixels at right of
chart }
```

## custom Components الباب العشرون : إنشاء

```
yScaleMax = 100.0;    { Maximum Y scale value }
yScaleIncrement = 10.0; { Increment for Y scale
markers }
{ Typed constants }
spaceVertical: Integer = spaceAtTop + spaceAtBottom;
spaceHorizontal: Integer = spaceAtLeft +
spaceAtRight;
yScale: Integer = Trunc(yScaleMax / yScaleIncrement);
{ Array of colors used to draw topmost bars }
colorArray: array[0 .. numClrs _ 1] of TColor = (
    $0000000, $0FFFFFFF, $0FF0000, $000FF00,
    $00000FF, $0FFFF00, $000FFFF, $0FF00FF,
    $0880000, $0008800, $0000088, $0888800,
    $0008888, $0880088, $0448844, $0884488
);

{ Return Y-coordinate for data point N }
function TBarChart.YData (N: Integer): Integer;
var
    F: Double;
begin
    F := (StrToFloat (FData [N]) / yScaleIncrement) *
        yIncrement;
    Result := YBase _ Round(F);
end;

{ Paint component shape at runtime AND design time }
procedure TBarChart.Paint;
var
    XMax, YMax: Integer;
    Width1, WidthD2: Integer;
    I, X1, Y1, X2, Y2: Integer;
begin
    with Canvas do
        begin
            { Erase background }
```

```

Pen.Color := FPen.Color;
Brush.Color := FBrush.Color;
X1 := Pen.Width div 2;
Y1 := X1;
XMax := Width _ Pen.Width + 1;
YMax := Height _ Pen.Width + 1;
Rectangle(X1, Y1, X1 + XMax, Y1 + YMax);
if FData.Count = 0 then Exit;
{ Initialize variables }
try
  XIncrement := (XMax _ spaceHorizontal) div
    FData.Count;
  YIncrement := (YMax _ spaceVertical) div
    yScale;
  Width1 := XIncrement div 2;
  WidthD2 := Width1 div 2;
  XBase := spaceAtLeft + WidthD2;
  YBase := YMax _ spaceAtBottom;
  Canvas.Font := Self.Font;
{ Draw bargraph }
  for I := 0 to FData.Count _ 1 do
    begin
      X1 := spaceAtLeft + (XIncrement * I);
      Y1 := YData(I);
      X2 := X1 + Width1;
      Y2 := YBase;
      if FLabels then
        begin
          Brush.Color := FBrush.Color;
          TextOut(X1, Y1 _ 30, FData.Strings[I]);
        end;
      end;
      Brush.Color := clBlack;
      Rectangle(X1 + 4, Y1 _ 4, X2 + 4, Y2 _ 4);
      Brush.Color := colorArray[(I + 2) mod
        numClrs];
      Rectangle(X1, Y1, X2, Y2);
    end;
  end;

```

```

end;
except
  ShowMessage('Error in data point ' + IntToStr(I));
  FData.Clear;
  Invalidate;
end;
end;
end;

```

وبالطبع ، يحتاج ال component الخاص بك أن يقوم بما هو أكثر من مجرد عرض مربع بسيط . إن باقى ال TBarChart Paint method يرسم لل bars و text labels (إذا كانت ال FLabels محددة بـ True) . وهذا الجزء من البرنامج يغطى عنصرين هامين . الأول ، إذا كانت ال FData.Count مساوية للصفر ، فلا يوجد نقاط بيانات فى قائمة ال string لخاصية ال Data . ويتهى ال Paint فى هذه الحالة ليمنع خطأ القسمة على صفر . والجانب الثانى هو معالجة ال exception ، والذي يوفره ال Paint بإدخال أوامر الرسم والحساب الخاصة به فى try block .

وبالرغم من هذا المستوى من الحماية ، فإن أى exceptions تحدث فى ال Paint method يمكن أن تؤدى الى loop لا نهائية- حالة لا قياسية بأن ال TBarChart تعمل فى ال except block الخاص بها . اذا وقع خطأ فى ال try block ، يعرض ال Delphi ال displays والذي عندما يكون مغلقاً ، يجعل النافذة السفلية تصبح غير صالحة ، مما يجعل ال Windows يصدر رسالة wm\_Paint أخرى ، مما يعيد إدخال ال Paint method ، والذي يولد exception آخر ، وهكذا الى ما لا نهاية .

لهذا السبب ، تعالج ال TBarChart أى exceptions بعرض رسالة خطأ ، أو مسح ال FData string list ، وإدخال صلاحية النافذة . جرب هذا بإدخال بعض البيانات مثل XYZ أو string آخر فى خاصية ال Data . سوف تتلقى رسالة خطأ تخبرك بفهرس نقطة البيانات الخطأ (إن الفهرس الأول هو صفر كما هو الحال فى كل string lists) ، ويصبح ال object خالياً .

وبدلاً من ال clear Data ، وهو غالباً ليس أفضل استجابة ، قد يعرض البرنامج رسالة أو رمز داخل ال object ، أو يمكن أن يحدد علماً يمكن للبرنامج أن يفحصه . قد تريد تجربة هذه الاستجابات البديلة فى ال except block .



## component property access methods

توضح القائمة (٦-٢٠) باقى برمجة ال TBarChart . وهذه ال  
procedures تنفيذ methods وصول الخاصية لحقول ال FPen ، FBrush ،  
FLabels و FData .

**القائمة (٦-٢٠)، تنفيذ method وصول لخاصية ال TBarChart**  
 { Local event handler redraws shape when necessary }  
 procedure TBarChart.StyleChanged(Sender: TObject);  
 begin  
     Invalidate;  
 end;  
  
 { Assign new brush data to FBrush field }  
 procedure TBarChart.SetBrush(Value: TBrush);  
 begin  
     FBrush.Assign(Value);  
 end;  
  
 { Assign new pen data to FPen field }  
 procedure TBarChart.SetPen(Value: TPen);  
 begin  
     FPen.Assign(Value);  
 end;  
  
 { Assign new string list to FData field }  
 procedure TBarChart.SetData(Value: TStrings);  
 begin  
     FData.Assign(Value);  
     Invalidate;  
 end;  
  
 { Assign new Boolean value to FLabels field } .  
 procedure TBarChart.SetLabels(Value: Boolean);  
 begin

## الباب العشرون : إنشاء custom Components

```
if FLabels <> Value then { Exit if no change needed }
begin
  FLabels := Value; { Assign to FLabels NOT Labels
  !}
  Invalidate; {Redraw component to add/remove
  labels}
end;
end;

end.
```

إن الـ `StyleChanged` procedure يستدعي الـ `Invalidate` لذا يصدر الـ `Windows` رسالة `wm_Paint` لنا لنافذة الـ `object`. هذا يؤدي في النهاية إلى استدعاء الـ `Paint` والذي يحو ويعيد رسم صورة الـ `bar chart`. إن البرنامج لا يقوم باستدعاء الـ `StyleChanged`؛ فهو يعين الـ `procedure` للـ `OnChange` للـ `TPen` والـ `TBrush`. ولذلك، فإن أية تغييرات لخصائص الـ `Pen` والـ `Brush` تؤدي لإعادة رسم الـ `object`.

وتعمل الـ `SetBrush Procedures` والـ `SetPen` والـ `SetData` بطريقة متشابهة. فكلًا منها يتلقى `parameter` من نوع الـ `field` والذي يقومون بتمريره للـ `Assign method` للحقل. يمكنك أن تثق من أن الـ `Assign` يفعل كل ما هو ضروري للتخلص من أى `resources` موجودة قبل قبول البيانات الجديدة. على سبيل المثال، إن تعيين `Pen` جديد يتخلص من الـ `pen resource` للـ `Windows` (إن وجد) المشار إليه بواسطة حقل الـ `FPen`.

لاحظ أن الـ `SetData`، الذي يعين خاصية الـ `Data string list`، يقوم باستدعاء الـ `Invalidate` لإعادة رسم النافذة. وهذا لأن الـ `TStrings class` لا توفر `OnChange`؛ وإذا كانت توفره، يمكنك تعيين الـ `StyleChanged` كما كان الحال مع خصائص الـ `Pen` والـ `Brush`.

في `method` الوصول لخاصية، يمكن أن يساعد التأكد مما إذا كانت قيمة ما هي نفسها كالقيمة الجديدة على منع الـ `flicker`، وخاصة للـ `components` البصرية المعقدة مثل الـ `BarChart`. إذا قامت الـ `SetLabels` بتعيين القيمة الجديدة واستدعت الـ `Invalidate`، فسوف يحدث للـ `object` الـ `flutter` في كل مرة يقوم المستخدمون فيها بإبراز خاصية الـ `Labels` في الـ `Object Inspector`.

تعين الـ SetLabels قيمة مستخدم لفتح الـ Boolean للـ FLabel. وهذا الـ procedure يوضح تصميمياً عاماً لـ methods الوصول لخاصية. وطبقاً للمعروف، تقوم الـ method أولاً بالتأكد من أن الـ Value هي نفسها مثل حقل الـ object. إذا كانت كذلك، فلا يمكن فعل شيء، ويتهى الـ procedure. وإن لم تكن، يعين البرنامج Value لحقل الـ FLabels، ويطل صلاحية النافذة لإعادة رسمها وإزالة أو إدخال labels نقاط البيانات.

لا تقم ابداً بتعيين قيم لخصائص في الـ method الوصول لخاصية. وغين قيم فقط لحقول البيانات، والمعرفة غالباً في القطاع الـ private في الـ class. ان تجاهل هذه القاعدة يمكن ان يؤدي الى زيادة المخزون، وقد يغلق Delphi والـ Windows. في القائمة النموذجية، تعتبر الـ SetLabels هي الـ method الوصول الذي يستدعيه البرنامج للتعليقات لخاصية الـ Labels. اذا كانت الـ SetLabels لتعيين قيم للـ Labels، وكان البرنامج قد استدعى في الـ SetLabels مراراً وتكراراً الى ان ينفجر المخزون. كن شديد الحذر عند كتابة method وصول لخاصية لتعيين قيم لحقول بيانات فعلية فقط في الـ class.

**تحذير:** إن تعيين قيم لخصائص في الـ method الوصول لخاصية يمكن أن يؤدي الى ما يسمى بالتكرار اللانهائي، مما يؤدي الى زيادة المخزون. وبالطبع، إن التكرار لا يكون أبدى لأنه يؤدي الى إيقاف النظام، ولكن لا عليك بالتسمية فقط لا تفعل هذا!!



## فهم تصميم الـ Component:

إن ما تبقى من هذا الباب يقدم أفكاراً تساعد على فهم تقنيات تصميم الـ component. كما ذكرت، إن إنشاء الـ component موضوع قد يملأ كتاباً بهذا الحجم، لذا فإن ما يلي لا يعتبر كاملاً ووافياً. ولكن، لقد حاولت ان اتلمس الموضوعات التي توليها اهتماماً رئيسياً كمؤلف للـ component، والتي تقترح موضوعات للبحث في مصادر اخرى.

## الـ Custom components:

في الـ VCL، تمر على components تحمل الكلمة Custom في أسمائها. على سبيل المثال، ان الـ TCustomEdit مشتق من الـ TWinControl في الـ StdCtrls unit باستخدام هذا التعريف:

## الباب العشرون : إنشاء custom Components

```
TCustomEdit = class(TWinControl)
```

```
...
```

```
end;
```

وبعد ذلك مباشرة، نجد هذا التعريف لل TEdit component، وهو ال component الذى يستخدمه التطبيق :

```
TEdit = class(TCustomEdit)
```

```
...
```

```
end;
```

والسبب فى وجود two classes بدلاً من مجرد ال TEdit هى ان نرودك بـ raw edit-control class بلا خصائص ال published (فى الواقع، هناك خاصية واحدة، وهى ال TabStop، والتي لها قيمة افتراضية هى True) وال Components التى تحمل كلمة Custom فى اسمائها لا تجعل الخصائص ال published. وقد تحتوى ايضاً على تعريفات لل abstract method والتي تتوقع ال class المشتقة تنفيذها.

اذا قمت باشتقاق ال TEdit، فإن كل الخصائص ال published لها تصبح ال published فى ال class المشتقة. واذا قمت، بدلاً من ذلك، باشتقاق class من TCustomEdit، والتي توفر البرمجة الفعلية لل TEdit، فإنك تحدد أى الخصائص يتم نشرها. وتنفذ ال class المشتقة ايضاً أى abstract methods.

### :Component debugging

ان إزالة الاخطاء واختبار ال components يصبح امراً معقداً بسبب انك تستخدم نفس نظام التطوير الذى تريد ان تثبت ال components فيه. يستطيع Delphi ان يقوم بعملية ال compile فقط لمشروعات لإزالة الاخطاء- لإزالة اخطاء component يجب ان تثبتها على ال VCL وتكتب تطبيقاً اختبارياً. عندما لا يكون هذا ملائماً، يمكنك ان تنتهج أحد اسلوبيين :

\* قم بتطوير ال component على انه تطبيق، ثم قم بتحويله الى component. لقد انشأت ال TBarChart باستخدام هذا ال method. ان البرنامج الاصلى، وهو غير مذكور هنا، قد عرض bar chart باستخدام class

مشتقة من الـ TGraphicControl . ثم قمت بنسخ هذه الـ code فى الـ component unit. النهائية.

\* قم بضم component unit. فى مشروع اختبارى، وقم بإنشاء objects من الـ component class فى procedure أو function. يجب عليك ان تقوم بتشغيل البرنامج لترى الـ component، ولكن هذه الطريقة تساعدك على إزالة الاخطاء من الـ class قبل تثبيت الـ component على الـ palette.

**على القرص المدمج:** ولتجربة التقنية الثانية- والتي تعتبر ناجحة فى إنشاء واختبار الـ component المعقدة- انسخ ملف الـ BarChart.pas من على دليل الـ Source\BarChart بالقرص المدمج الى دليل جديد وخالى. اتبع هذه الخطوات لإنشاء برنامج اختيارى ينشئ BarChart تحت التحكم الكامل للبرنامج. توضح القائمة (٢٠-٧) كيفية إنشاء object فى وقت التشغيل لـ component ليس موجوداً على لوحة الـ component.



١- إبدأ تطبيقاً جديداً. إجعل اسم الـ form MainForm، وحدد الـ Caption الخاص بها بـ Test BarChart Component. احفظ المشروع فى نفس الدليل الذى يحتوى على نسخة من الـ BarChart.pas. (من الناحية العملية قد تختلف الأدلة، ولكن من الأسهل أن تستخدم نفس الدليل لهذا العرض). اجعل اسم الـ unit Main.pas واسم المشروع BarTest.pas.

٢- اضع BarChart الى أمر الـ uses بالـ Main unit.

٣- بدلاً من استخدام الـ BarChart على الـ VCL (عند اختبار component جديد، فهذا لن يكون ممكناً)، قم بتعريف object من الـ BarChart class يدوياً. افعل هذا بكتابة تعريفات الـ object فى الـ class form- تماماً كما يفعل Delphi بصورة تلقائى للـ components النامة. على سبيل المثال، أدخل التعريف التالى فى القطاع الـ public للـ TMainForm class:

```
TMainForm = class(TForm)
private
    { Private declarations }
public
```

### BarChart1: TBarChart;

end;

٤- قم بإنشاء الـ OnCreate الخاص بالـ form (أو يمكنك استخدام أى component -event handler للـ Button ، مثلاً). قم بإنشاء component instances التى تحتاجها. هذا يتطلب على الأقل خطوتين : (١) استدع Create لإنشاء الـ object ، و (٢) عين parent object لحقل الـ parent object . وغالباً ، تستطيع ان تستخدم الـ Self كـ arguments فى كلتا المهمتين . وتشير الـ Self الى الـ form object (لان هذا يعد method للـ form class) ، وإنك غالباً ما تريد ان تمتلك الـ form object الناتج من الخطوة رقم (١) وتكون الاساس لها [الخطوة رقم (٢)]. على سبيل المثال ، استخدم هذه العبارات لإنشاء BarChart :

BarChart1 := TBarChart.Create(Self);

BarChart1.Parent := Self;

٥- يجب عليك ايضاً ان تبدأ حقول اخرى مثل Left ، Right ، Width ، و Height ، التى تحدد موضع الـ object وحجمه . بالإضافة الى ذلك ، إنك تحتاج لـ إضافة عبارات لتعيين أية بيانات مطلوبة -قائمة الـ Data string للـ BarChart ، مثلاً . توضح القائمة (٢٠-٧) الـ unit التامة ، والمخزنة على انها Main2.pas فى دليل الـ Source\BarChart على القرص المدمج . ويوضح هذا الملف كيفية إنشاء object فى وقت construct ليس على VCL . استخدم هذا الملف لإنشاء مشروع اختبارى للـ BarChart class .

القائمة (٢٠-٧) : Barchart\Main2.Pas . استخدم هذا الملف لإنشاء مشروع

اختبارى للـ BarChart class

unit Main;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics,  
Controls,  
Forms, Dialogs, BarChart;

```

type
  TMainForm = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    BarChart1: TBarChart;
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FormCreate(Sender: TObject);
begin
  BarChart1 := TBarChart.Create(Self);
  BarChart1.Parent := Self;
  with BarChart1 do
  begin
    Left := 20;
    Top := 20;
    Width := 375;
    Height := 200;
    Data.Add('65.0');
    Data.Add('45.0');
    Data.Add('95.0');
    Data.Add('76.0');
    Data.Add('51.0');
    Data.Add('90.0');
  end;
end;

end.

```

لا تقم بادخال حقول بيانات أو تعريفات للـ method فى قطاع الـ published الافتراضية للـ class form (ان القطاع التالى مباشرة لتعريف الـ class حتى محدد الوصول، وغالباً يكون القطاع الـ private). اذا قمت بتعديل القطاع الـ published الافتراضى، فإنك تتلقى رسالة خطأ لأنه هذا هو المكان الذى يعرف فيه Delphi الـ objects والـ event handlers.

### خصائص الـ Class:

ان الخاصية قد تكون أى نوع بيانات فيما عدا الملف. ولكن، لا يمكن للخاصية ان تكون Pascal array، لان الخصائص ليست حقول حقيقية والوصول إليها محكوم بـ methods القراءة والكتابة. ولكن، يمكنك انشاء خصائص شبيهة بالـ array التى تستدعى methods لقراءة وكتابة عناصرها. (لمزيد من المعلومات، انظر خصائص الـ array فى هذا الباب). باستثناء هذه القيود، قد تنشر الخاصية أى نوع من الحقول.

ان الـ events تعد حقول بيانات فعلية للـ TNotifyEvent أو نوع مشابه. ان الـ event هو pointer لـ method ثم توفيره من قبل المستخدم. ولكن الـ event فى حد ذاته method، ولذلك، فإن نشر الـ events يؤدى الى نشر حقول بيانات (الـ pointer)، ليس code.

تظهر الخصائص لمستخدم الـ component كحقول، ولكنها methods مغلقة لقراءة وكتابة حقول بيانات اخرى، والتى تعتبر خاصية للـ component class. ويمكن أيضاً للخصائص ان تكون محسوبة، على سبيل المثال، يمكنك كتابة method الوصول الخاصية وتحسب قيمة الخاصية من حقول بيانات اخرى. ولكن، كن على حذر من ان تقدم الكثير من هذه الـ methods فيما بين الخصائص المحسوبة، أو يصبح ترتيب بدءها فى وضع حساس.

ان نشر الخصائص يجعل معلومات وقت التشغيل متاحة عن الـ class التى تستخدمها Delphi بعدة طرق. على سبيل المثال، يستخدم الـ Object Inspector معلومات نوع لعرض وتحرير قيم الخاصية، وهى مخزنة فى ملف الـ .dfm. بعد compile المشروع الـ BarTest لهذا الباب، افتح ملف الـ Main.dfm لتستعرض كيف يقوم Delphi بتخزين الخصائص. توضح القائمة (٢٠-٨) ملفاً. استخدم Delphi لفتح الملف لفحص الخصائص.



```

القائمة (٢٠-٨): ملف الـ BarchartMain.dfm لشروع الـ BarTest
object MainForm: TMainForm
  Left = 200
  Top = 95
  Width = 435
  Height = 300
...
object BarChart1: TBarChart
  Left = 24
  Top = 16
  Width = 377
  Height = 209
  Hint = 'BarChart component'
  Brush.Color = clSilver
  Data.Strings = (
    '65.0'
    '45.0'
    ...
    '90.0')
  Labels = True
  ParentShowHint = False
  ShowHint = True
end
end

```

هذا يبدو كالـ Pascal، ولكنه ليس كذلك. انه ملف يحتوى على قيم الخواص. إن ملف الـ form يحتوى على قيم خاصية كتلك التى تدخل فى نافذة الـ Object Inspector. يقوم Delphi بتحميل قيم الخاصية بعد أن يستدعى constructor الـ component، والذي يقوم ببدء قيم الخاصية.

إن قيم الخاصية يمكن إنشائها بأكثر من طريقة. بعد استدعاء الـ component constructor وتحميل قيم الخاصية من ملف الـ form (أو صورته المرتبطة بملف code الـ .exe)، يقوم Delphi باستدعاء Loaded method ظاهرى والذي يمكنك السيطرة عليه. ضع هذا التعريف فى القطاع الـ protected للـ component class.

## الباب العشرون : إنشاء custom Components

procedure Loaded; override;

قم بتنفيذ ال procedure لأداء أى إنشاءات إضافية لـ object. وهذه البدايات تأخذ الأولوية عن أى قيم معينة فى ال constructor أو من ملف ال form (بمعنى آخر ، بواسطة ال Object Inspector). لا يزال المستخدمون قادرون على تحديد خصائص باستخدام ال Object Inspector ، ولكن عندما يقوموا بتشغيل البرنامج ، يتم استدعاء ال Loaded مرة أخرى ، لذا لا تستبدل أى قيم تريد أن يتوصل إليها المستخدمون .

ويتعبر ال Loaded مفيداً فى توفير قيم افتراضية للخصائص التى يقرر المستخدمون أن يتركوها فارغة أو التى ليس لها قيم ملف خاصية . على سبيل المثال ال Loaded function ، إن ال DBNavigator يستخدم هذه التقنية لإنشاء hint-text strings إشارة افتراضية اذا لم يدخل المستخدم أى strings فى حقل ال Hints .

استدع دائماً ال Loaded method الموروث فى ال procedure الاستبدال الخاص بك ، ويعتبر ال Loaded أيضاً هو الفرصة الوحيدة التى تحصل عليها للوصول الى أى objects ملحقة خلال عملية التحميل . على سبيل المثال ، لا يستطيع ال data-ware component استخدام خاصية ال DataSource التابعة له حتى يتم استدعاء ال Loaded .

**ملحوظة:** إن ال Loaded method لا يعد جزءاً من خطوات إنشاء ال component ولهذا السبب لا يجب أن تضع code هامة فى Loaded method موروث لأنه لا يتم استدعاؤه اذا كان ال component يتم إنشاؤه ديناميكياً فى وقت التشغيل بدلاً من تحميل ال from ك form stream .

Note

### خصائص ال Array:

من الممكن ألا تكون الخصائص Pascal arrays ، ولكن يمكنك إنشاء خاصية array يتم استخدامها تماماً مثل ال array . وتعتبر خصائص ال array قيمة وبخاصية فى إنشاء هياكل بيانات متقدمة مثل ال arrays المرتبطة والموزعة . ومثل جميع الخصائص ، توفر خاصية المتجة methods وصول لقراءة وكتابة البيانات .

ويمكن لـ class الخاصية ان تختار أى method للتخزين الفعلى البيانات . على سبيل المثال ، هذه هى الطريقة التى توفر بها خاصية الـ TStringList وصولاً لبيانات الـ string باستخدام فهرسة الـ array ، ولكن داخلياً ، تربط الـ strings ببراعة فى list .

وقد تستخدم خاصية الـ array ايضاً أى نوع من نوع البيانات للفهارس . يمكنك إنشاء array مفهرس على strings أو قيم النقطة القائمة . ان الـ Pascal arrays العامة قد تستخدم فقط فهارس تركيبة مثل الاعداد الصحيحة ، الرموز ، والثوابت المحدودة .

**على القرص المدمج:** ان القائمة (٩-٢٠) هى تطبيق DOS-prompt الذى يوضح كيفية برمجة خصائص الـ array . يمكنك ايضاً استخدام نفس التقنية فى الـ component ، ولكن كما يوضح هذا البرنامج ، يمكنك ايضاً نشر خصائص الـ array على انها اعضاء protected لأى class ويوجد هذا البرنامج على القرص المدمج فى دليل Source\Daynames ، فى ملف ArrayP.pas . لتشغيل البرنامج ، اضغط زر الـ Windows Start ، واختر Programs\MS DOS Prompt ؛ ثم تحول الى دليل الـ Daynames ، وإدخل . Arrayp



**القائمة (٩-٢٠) : Daynames\ArrayP.pas**  
 program ArrayProperties;

uses SysUtils;

type

```
TDayNames = class
private
    function GetName(N: Integer): String;
protected
    property DayStr[N: Integer]: String read GetName;
default;
end;
```

```
function TDayNames.GetName(N: Integer): String;
begin
  if (N < 0) or (N > 6) then
    raise ERangeError.Create('Array index out of range');
  case N of
    0: Result := 'Sunday';
    1: Result := 'Monday';
    2: Result := 'Tuesday';
    3: Result := 'Wednesday';
    4: Result := 'Thursday';
    5: Result := 'Friday';
    6: Result := 'Saturday';
  end;
end;

var
  DayNames: TDayNames;
  I: Integer;

begin
  DayNames := TDayNames.Create;
  try
    Writeln('Default property (DayNames[I])');
    for I := 0 to 6 do
      Writeln(DayNames[I]);
    Writeln;
    Writeln('Named property (DayNames.DayStr[I])');
    for I := 6 downto 0 do
      Writeln(DayNames.DayStr[I]);
    finally
      DayNames.Free;
    end;
  end.
end.
```

**ملحوظة:** لمزيد من المعلومات عن استخدام الـ DOS-prompt compiler خط الأمر لـ Delphi، انظر الباب الحادى والعشرين.

Note

ينشئ البرنامج class صغيرة، وهى TDayNames، والتى تقدم ايام الاسبوع فى شكل pseudo-string array. الـ class تعرف function خاصة، وهى الـ GetName، والتى يتم استخدامها على انها method وصول للقراءة للـ pseudo-string array. ويعرف القطاع الـ protected للـ class لـ method الوصول للخاصية، DayStr، متبوعاً باقواس مربعة تحتوى على argument عدد صحيح. وهذه الـ argument يمكن ان يكون أى نوع بيانات، ولو حتى string. ويكون قوس الإغلاق متبوعاً بنقطتين، نوع بيانات عناصر الـ array، و method الوصول للقراءة، وهو GetName.

ان إنهاء تعريف الخاصية بـ default يحدد هذه الخاصية على انها الخاصية الافتراضية للـ class. هذا يعنى ان المستخدمين لـ class objects يمكن ان يعاملوا الـ objects على انه الخاصية. ويمكن ان يكون خاصية واحدة فقط لأى class هى البديل الافتراضى. على سبيل المثال، ان الـ for loop الاولى للبرنامج تستخدم الـ DayNames object وكأنه array:

```
for I := 0 to 6 do
```

```
  Writeln(DayNames[I]);
```

يمكنك أيضاً الإشارة الى الخاصية على انها حقل فى الـ object:

```
for I := 6 downto 0 do
```

```
  Writeln(DayNames.DayStr[I]);
```

وتستخدم الـ TStrings class نفس هذا الـ method لتوفير وصولاً الى strings باستخدام تعبير مثل Items[I] أو Items.Strings[I]، وهما نفس الشئ لان الـ Strings هى الخاصية الافتراضية للـ TStrings.

عندما يستخدم البرنامج تعبيراً مثل DayNames[I]، يولد الـ compiler استدعاءً للـ method الوصول الى القراءة المحدد، وهو فى هذا المثال، الـ function GetName. وفى القائمة، تقوم الـ GetName أولاً بالتأكد مما اذا كان

## الباب العشرون : إنشاء custom Components

ال فهرس فى النطاق- اذا لم يكن كذلك ، فإنها تحدث ERangeError exception  
إدخلى هذه العبارة فى ال try block للبرنامج الرئيسى لأختبار ال exception :

```
WriteLn(DayNames[7]); { ??? }
```

وتستخدم ال Function GetNames عبارة case لإدخال string اسم يوم  
لقيم الفهرس من صفر الى ستة ولان ال class للقراءة فقط ، فإنها توفر method  
وصول للكتابة ، ولكن هذا امر يسهل القيام به . قم بتعريف method وصول الى  
الكتابة ، والذي يجب ان يكون له parameter قيمة فهرس (يمكن ان يكون أى  
نوع ، ولكن هنا فهو Integer) ، كما يلى :

```
procedure SetName(N: Integer; const S: String);
```

بعد ذلك ، اصف SetName الى تعريف الخاصة :

```
property DayStr[N: Integer]: String read GetName write  
SetName; default;
```

قم بتنفيذ ال SetName procedure لتخزين string الذى تم تمريره فى  
الفهرس المحدد . على سبيل المثال ، يمكنك إدخال ال string فى قائمة وتسجيل قيمة  
الفهرس فى object ما . سوف يبحث method الوصول الى القراءة فى القائمة عن  
هذا الفهرس ويدخل ال string المرتبط به- وهذا يعد مثلاً على ال array المرتبط  
والموزع ان ال string المدخلة فقط هى التى يتم تخزينها على قائمة ، ومواضع  
الفهرس الغير مستخدمة فلا تشغل مساحة .

وال non components classes مثل ال TDayNames يمكن الا تنشئ  
خصائص ، ولكن يمكن أن تعرف فى قطاع protected أو public كما هو موضح  
هنا . وهذه تقنية مفيدة فى التحكم فى الوصول الى حقول البيانات .

## إنشاء ال ActiveX Controls:

يمكن تحويل أى Delphi component بسهولة الى ActiveX control .  
وهذا ال control يمكن استخدامه عندئذ فى تطوير نظم مثل ال Visual Basic ،  
Microsoft Access ، Paradox لل Windows ، والبرمجيات الأخرى التى  
تؤيد بروتوكول ال ActiveX . يمكنك أيضاً استخدام ال ActiveX الخاصة بك فى

Delphi وفى ال C++ Builder الخاص بال Borland، بالرغم من انه اذا كان لديك ال control الأصيل ك component، فإن افضل استخدام يكون فى ال form الأصلية فى Delphi.

ك ActiveX control، فإن Delphi component يكون محاطاً فى غلاف الذى يحكم الاتصال بين ال control والعالم الخارجى. ومواجهة التطبيق هذه تتفق مع تخصيصات ال ActiveX لـ Microsoft، وهى لا تختلف عن أى ActiveX آخر. ولكن، ال control يحتوى ال Delphi component داخله.

### كيف تنشئ ال ActiveX:

لإنشاء ActiveX هناك خطوات عديدة:

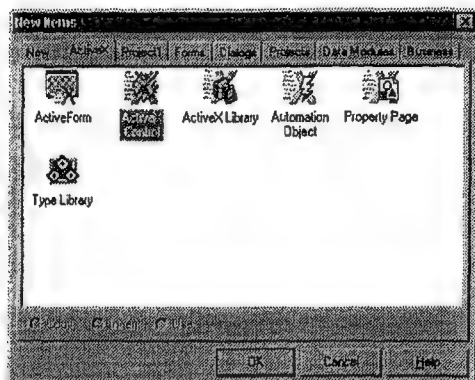
- ١- قم بتصميم واختيار component كما هو موضح فى هذا الباب.
- ٢- قم بتشغيل ال ActiveX Control Wizard الخاص بـ Delphi لنحول ال component الى ال ActiveX control.
- ٣- واختيارياً، قم بتشغيل ال ActiveX Property Page Wizard. هذه الخطوة تؤدي الى إنشاء صفحة خاصية والتى تمكن المستخدمين من استعراض وتحرير خصائص ال control والتى تعتبر ضرورية اذا كان كل ما تريده هو نشر ال control على سبيل المثال، المتصفح ال Internet. اذا اخذت هذه الخطوة، فإنك ايضاً تربط صفحة الخاصية بال ActiveX control.
- ٤- بعد ال compil يتم تسجيل ال control مع ال Windows. يجب ان يتم تسجيل كل ال ActiveX قبل ان يتم استخدامها.

عندما تقوم بإنشاء ActiveX من component، يقوم Delphi بصورة تلقائية بإنشاء type librar والتى تصف واجهة تطبيق ال control. وهذه المكتبة، والمخزنة فى ملف .tlb، تستخدمها نظم التطوير للوصول الى خصائص ال control، وال methods، وال events. تحتوى المكتبة أى control آخر أو ملف code يستخدم ال ActiveX عندما تقوم بـ compile هذا ال control أو التطبيق.

## تحويل ال DingButton الى ActiveX

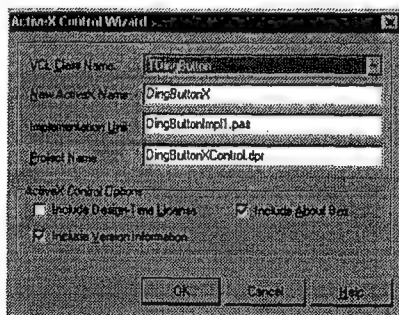
إتبع هذه الخطوات لتحويل ال DingButton الخاص بهذا الباب الى  
ActiveX :

١- اختر File|New... واختر باب صفحة ال ActiveX من ال New  
Items dialog [انظر شكل (٢٠-٣)].



شكل (٢٠-٣): هذه هي ال ActiveX في ال New Items dialog

٢- اضغط مرتين ايقونة ال ActiveX Control. هذا يؤدي الى عرض  
ال ActiveX Control Wizard dialog الموضوع في شكل (٢٠-٤).



شكل (٢٠-٤): اختر ال component لتحويله الى ActiveX

٣- اختر ال component لتحويله الى ActiveX من على قائمة ال VCL  
Class Name. عندما تفعل هذا، يملأ Delphi الحقول الاخرى، والتي يمكنك  
تحريرها اذا اردت. على سبيل المثال، يمكنك إدخال ActiveX Name



و Implementation Unit مختلف، والذي يحتوى على source code لغلاف واجهة تطبيق الـ ActiveX الخاص بالـ component. اضغط Ok عندما تنتهى.

**ملحوظة:** ان الـ ActiveX يجب ان تصنف كجزء من مشروع مكتبة الـ ActiveX. فى هذه النقطة، يحذرك Delphi اذا لم يكن لديك مثل هذه المكتبة مفتوحة- اجب بـ Yes لإنشاء مكتبة جديدة وأكمل العمل.

Nota

ان الخطوات (٢) و (٣) تقدم ثلاثة اختيارات يمكنك تشغيلها (انظر check boxes فى اسفل نافذة الـ dialog). وهذه الخيارات هى:

#### • Include Design-Time License

وهذا يؤدى الى إنشاء ملف lic له مفتاح مطلوب لفتح الـ control للاستخدام فى بيئة تصميم البرمجيات. تأكد من هذا الخيار اذا كنت تريد ان تقيّد استخدام الـ control الخاص بك بالنسبة للمستخدمين النهائيين (كما هو الحال فى متصفح الـ Internet)، ولكنك لا تريد مطورى البرمجيات أن يحتوى الـ control فى برامجهم دون الحصول على رخصة وقت التصميم منك.

#### • Include Version Information

هذا يضيف معلومات نسخة للملف الـ .ocx الخاص بالـ control. يستطيع المستخدمون ان يروا هذه المعلومات. لإدخال معلومات نسختك، اختر ProjectOptions... واختر باب صفحة الـ VersionInfo.

#### • Include About Box

وهذا يؤدى الى إنشاء الـ about-box dialog unit، والتي يتم إضافتها لمشروع مكتبة الـ ActiveX. يمكنك تحرير الـ about-box dialog هذا كما تفعل مع أى Delphi form أخرى. على سبيل المثال، اختر الـ about-box dialog فى نافذة الـ code editor واضغط F12 لإظهار الـ form البصرية المرتبطة به. يمكنك عندئذ ان تضيف Buttons و Labels و components أخرى إلى الـ about-box dialog، والذي يتم عرضه فى بيئة التطوير الخاصة بالمستخدم.

بعد اختيار الخيارات المتنوعة المتاحة لإنشاء الـ ActiveX، يمكنك تحرير وبرمجة الـ units الناتجة فى مشروع المكتبة. عندما تنتهى، اختر ProjectBuild

...all لها compile ال control . هذا يؤدي الى إنشاء ملف ocx . على القرص الذي يحتوي على ال ActiveX .

### استخدام ال ActiveX :

بعد بناء ال ActiveX ، يجب ان تقوم بتسجيله مع ال Windows لتجعل ال control متاحة للتطبيقات الاخرى (بما فيهم Delphi) . وهذا أمر سهل - اختر امر ال RunRegister ActiveX Server . يجب ان يكون مشروع مكتبة ال ActiveX مفتوحاً حتى يمكن تشغيل هذا الامر .

يسجل Delphi ال control (أو ال controls) في مشروع مكتبة ال ActiveX الحالي . عندما ينتهي ، Delphi يعرض information dialog والذي يخبرك ان العملية قد تمت بنجاح . ان ال control متوفراً الآن للاستخدام ، ولكن ، لم يتم تثبيته بعد على لوحة Delphi . لتفعل هذا ، قم بخطوة واحدة إضافية .

**Tip** **فكرة:** لإزالة ال ActiveX control من النظام ، افتح ملف المشروع الخاص به واختر RunUnregister ActiveX Server . هذا يؤدي الى إزالة ال control من سجل ال Windows .

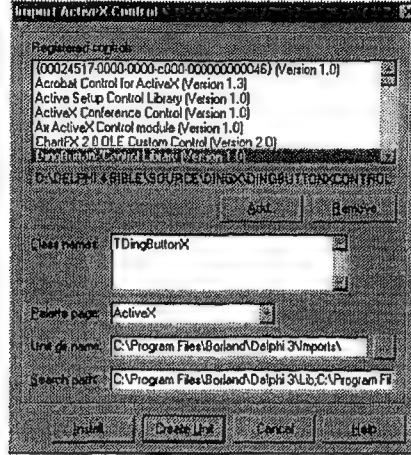
لإستخدام ال ActiveX component في Delphi ، اختر أمر ال ComponentImport ActiveX Control . هذا يؤدي الى إظهار ال Import ActiveX Control dialog الموضح في شكل (٢٠-٥) . ويذكر ال dialog كل ال ActiveX المسجلة مع ال Windows .

اختر واحداً من ال controls المسجلة لتدخله في Delphi . يمكنك اختيار ال palette page لتثبيت ال controls ، ولكن افضل ما يكون هذا في الصفحة الافتراضية ، ActiveX . اضغط زر ال Install لإدخال ال ActiveX في Delphi .

**ملحوظة:** ان الخطوات الموجودة في هذا الفصل هي لاغراض العرض الخالصة . ولان ادخال ActiveX في Delphi يؤدي بالفعل الى إنشاء غلاف Delphi component حول ال ActiveX ، فإنه من السخف ان يؤدي هذا العمل على ال ActiveX الذي هو في الحقيقة غلاف ملف حول ال Delphi component . وفي جميع الحالات ، من الافضل ان تستخدم

Note

Delphi component الاصلية مع Delphi. ولكن يمكنك استخدام الخطوات المشار إليها في هذا الفصل لإضافة ActiveX أخرى في Delphi للاستخدام ك component.



شكل (٢٠-٥)، ادخال ActiveX في Delphi باستخدام  
ال Import ActiveX Control dialog

## افكار للمستخدم الخبير

\* يمكنك توفير online help لل components الخاصة بك. عندما يختار المستخدمون component ويضغطوا F1، يبدأ Delphi البحث في online help مفهراً على class الكلمة الأساسية <name>، حيث أن <name> هي اسم ال class الخاص بك. لمزيد من المعلومات انظر العنوان التالي في online help "دمج ال help الخاصة بك في Delphi".

\* ان إزالة component من على VCL لا يؤثر على أى تطبيقات تستخدم هذا ال component. ولكن، يجب تثبيت ال component قبل ان تتمكن من تحميل أى مشروع يستخدمه.

\* لإنشاء خاصية تكون متاحة فقط في وقت التشغيل، ولكنها غير مرئية في نافذة ال Object Inspector، ضع تعريف الخاصية في القطاع ال public

## الباب العشرون : إنشاء custom Components

لـ class ، ولكن لا تنشره . (فى الواقع ، لا يمكن تعريف خاصية public على إنما published ، مما قد ينشئ مسارات وصول متضاربة الى الحقل المرتبط بها . ان الخاصية إما ان تكون ، public أو تكون published ؛ فلا يمكن ابدأ . ان تكون الاثنين معاً) .

\* يستطيع Delphi أن يقوم بالـ compile للمشروعات فقط . لـ compile unit تحت التطوير ، استخدمها فى مشروع اختبارى . يقوم Delphi بالـ compile للبيانات المعدله فى component units على حسب الحاجة عندما تقوم بتشيتها .

\* يمكن ان تكون الخصائص فى القطاعات الـ protected أو الـ published أو الـ public لـ class . يمكنك تعريف الخصائص فى القطاع الـ private لـ class ، ولكن لن يكون هناك جدوى من ان تفعل هذا لان class unit تستطيع الوصول الكامل لكل حقول البيانات الخاصة . ان غرض الخاصية هو توفير وصولاً محكماً للبيانات التى فى قطاعى الـ protected و الـ private .

\* يقوم الـ Object Inspector بصورة تلقائية بفرز اسماء الخاصية ايجدياً . يمكنك تعريف الخصائص بأى ترتيب فى الـ class . وهذا لا يؤثر على ترتيبها فى الـ Object Inspector .

\* تخزن الـ VCL الخصائص بحسب ترتيب التعريف ، ولكن يحدث التحميل فى أى ترتيب تكون عليه البيانات فى ملف الـ .dfm . لـ form . وهذا الترتيب يمكن ان يكون نفس ترتيب تعريف الخاصية ، ولكن يمكن ان يكون مختلفاً . على سبيل المثال ، يمكن للمستخدمين تحرير ملف الـ form كنص وتغيير قيم الخاصية وترتيبها . لا تقم ببناء components تعتمد على ترتيب تحميل الخاصية .

\* يمكن ان تكون الخصائص من أى نوع بيانات فيما عدا الملفات ، ولكن هناك قيود اضافية على انواع الخصائص التى يمكن ان لـ class أن نجعلها publish . لا يمكنك نشر الـ Pascal arrays ، مثلاً ، ولا يمكنك ايضاً أن تجعل خصائص القراءة فقط أو الكتابة فقط لان الـ Object Inspector

يجب ان تقرأها ويكتبها . ويمكن ايضاً للخصائص ال Published ال  
تستخدم أنواع بيانات ال Real أو ال Extended الخاصة بال Pascal  
الخصائص ال floating point، استخدم Single أو Double أو  
. Comp

\* ان packages وقت التشغيل القياسية يتم تسميتها بطريقة عشوائية نوعاً  
ما . على سبيل المثال، ان ال VCL40.bpl وال INET40bpl وال  
TEEDB40.bpl تعتبر جميعاً packages وقت تشغيل والطريقة الوحيدة  
لمعرفة هذه الحقيقية عن الملفات هى من خلال امتدادات اسماء ملفاتھا .

\* ان اغلب packages وقت التصميم يتم تسميتها بطريقة متوافقة . على  
سبيل المثال، ال DCLSTD40.bpl وال DCLDB40.bpl تعد  
packages وقت التصميم . وللأسف، فإن هذا التصميم يكسره ال  
IBEVNT40bpl، بالرغم من ان هذا packages ال  
IBEventAlerter component فى ال Samples palette، وكمثال،  
فإن استخدامه يكون "على مسئوليتك" . ان Borland لا يؤيد ال  
components ال sample ولقد تغيرت هذه ال component أو اختفت  
فى بعض الحالات على مدى حياة Delphi .

\* ان packages وقت التشغيل يتم تخزينها فى ال  
C:\Windows\System . وفيما عدا القليل من ال exceptions، يتم  
تخزين packages وقت التصميم فى دليل الخاص بـ \Bin الخاص بـ  
Delphi .

\* من المهم ان تفهم الفرق بين قائمة ال packages المطلوبة من قبل المستند  
الجديد وقائمة ال units التى يجب ان يتضمنها ال package الجديد .  
فالقاعدة الأولى تشير الى ال units الخارجية-على سبيل المثال، واحدة  
تحتوى على (subroutine) الذى تحتاجه وال package units الجديد .  
والقاعدة الثانية تشير الى ال units التى يجب ان يعمل عليها ال package  
الجديد . ويضم المستند هذه ال units، بنفس الطريقة تقريباً التى تضم بها  
ال class بياناتها وال code .

\* ان package وقت التصميم هو ببساطة package يذكر اسماء package وقت التشغيل الأخرى فى التعريف الخاص به .

\* ان compiles ال package يؤدى بصورة تلقائية الى compiles أى units من ال compiles بسبب التغير فى بيانات الملف الخاص بها (أى ، القديمة) منذ آخر مرة ثم فيها ال compile . لذلك قم دائماً بـ compile لـ package ان compile ال units منفردة لا يؤدى الى تحديثها فى ال package .



### المكتبروعات التى يمكنك تجربتها

(٢٠-١) : قم بإنشاء Date and Time والذي يمكنك إضافته على form أضف ميزة التنبيه الذى يدق جرساً أو يبعث رسالة . يمكنك استخدام حقل بيانات string-list لإدخال التواريخ والوقت المراد التنبيه عنها .

(٢٠-٢) : أضف خاصية ال Font لـ TBarChart . إنشئ ال Font لـ form font (انظر تعيين ال Paint method لـ Canvas.Font كإشارة حول كيفية فعل هذا) .

(٢٠-٣) : قم بتحسين معالجة ال exception الخاصة بالـ TBarChart . بدلاً من حذف نقاط البيانات اذا ما حدث exception ، جرب واحدة من الاستجابات المقترحة الأخرى . على سبيل المثال ، يمكنك عرض رمز قنبلة بدلاً من display ، أو يمكنك اختبار نقاط البيانات الفردية بشأن صلاحيتها بطريقة أخرى .

(٢٠-٤) : بإنشاء نسخة bitmap لـ RadioButton مع رمز مخصص (جوهرة ، مثلاً) فى مكان النقطة السوداء المعتادة .

(٢٠-٥) : متقدم . قم بإنشاء component أيقونة متحركة يعرض bitmaps متتالية . يمكنك استخدام ال component لعرض صور متحركة أثناء ملف ، أو طباعة ، أو عمليات أخرى طويلة .

(٦-٢٠): متقدم. أضف خاصية array للـ TBarChart class التى تنشر array اللون الثابت الحالى للـ unit. وهذا الـ array يوفر ١٦ قيمة من قيم الألوان لرسم bars متعاقبة. (للحصول على أكثر من ١٦ bars، تكرر الألوان). قم بتحسين الـ BarChart بجعله ممكناً للمستخدمين ليحرروا قيم الألوان هذه.

(٧-٢٠): قم بتحويل الـ BarChart الخاص بهذا الباب الى ActiveX.

### ملخص:

\* إن كتابة components خاصة بك ليس عملاً بسيطاً، ولكنه يمنح المبرمجين المتقدمين أدوات وإمكانات إضافية ليست متاحة لمطوري التطبيقات. يمكنك أن تؤسس الـ components الجديدة على أى من الـ components المتوفرة مع Delphi، أو يمكنك إنشاء components جديدة تماماً.

\* الـ component يعتبر unit، وتغريف class خاصة مع متطلبات Delphi للـ components البصرية. لتثبيت الـ component على لوحة الـ VCL، يجب إدخاله فى package. استخدم أمر الـ File|New... وعنصر الـ Packages لإنشاء package جديد.

\* تقوم الـ Components بنشر خصائص لتوفير methods الوصول لقراءة وكتابة حقول بيانات الـ class. يستطيع المستخدمون استعراض وتحرير الخصائص الـ published والتي لها methods وصول للكتابة فى نافذة الـ Object Inspector.

\* الـ Components تتطلب constructor methods و destructor الـ components البصرية بتنفيذ الـ Paint method لإنشاء مظهرها على الشاشة، فى كلاً من الـ form فى وقت التصميم، وفى البرنامج فى حالة تشغيله.

\* لإزالة الأخطاء من الـ components، قم بإنشائها تحت تحكم البرنامج بدلاً من تثبيتها على لوحة الـ component هذا يجعل من الممكن استخدام

## الباب العشرون : إنشاء custom Components

ال debugger الخاص بـ Delphi بتخطي برمجة ال component بخطوة واحدة لإدخال breakpoints ومتغيرات فحص .

\* يمكن أن تكون الخصائص أى نوع من أنواع البيانات فيما عدا الملفات وال Pascal arrays . يمكنك إنشاء خاصية array تقوم باستدعاء methods وصول للقراءة والكتابة ، ولكن هذه الخصائص يتم استخدامها كـ array . استخدم هذه التقنية لإنشاء هياكل بيانات array مرتبط وموزع .

\* يمكنك استخدام ال ActiveX Control Wizard لتحويل أى Delphi component الى ActiveX يمكنك أيضاً إدخال أى ActiveX فى Delphi أو استخدامه كـ component . ولكن ، من الأفضل دائماً أن تستخدم Delphi components الأصلية مع Delphi .

أتمنى أن تجد شيئاً ذا قيمة عالية فى الباب القادم ، والذي يجمع أفكاراً ، وحياً ، وتقنيات لشحن مهاراتك فى برمجة Delphi .

\*\*\*\*\*





# الباب الحادى والعشرين

## تطوير مهارات Delphi الخاصة بك

محتويات هذا الباب:

- Components
- ادوات Command-line
- تطبيقات CRT
- functions نافعة
- معلومات نوع وقت التشغيل
- Online help
- Dynamic Link Libraries
- event handlers للرسالة والتطبيق
- File streams
- تطبيقات Internet
- مزيداً من الافكار

بينما كنت اكتب هذا الكتاب ، جمعت العديد من الافكار والحيل والهامة لأقدمها فى هذا الباب الأخير . ان هذا الباب ليس له ترتيب أو هدف معين ، ولكن من المؤكد أنك ستجد العديد من الملاحظات والاقتراحات التى تساعدك على شحذ مهاراتك فى برمجة Delphi . فى هذا الفصل سوف نتعلم كيفية استخدام ادوات (command-line) النافعة ، وأفكاراً حول إنشاء واستخدام online help ،

وكيفية إنشاء واستخدام الـ dynamic link libraries ، وكيفية إدخال forms components أخرى فى الـ DLL ، وكيفية استخدام معلومات نوع وقت التشغيل . يمكنك أيضاً أن تجد معلومات حول الـ event handlers الخاصة بالتطبيق ، سير الملف ، التطبيقات الـ multithreaded ، مقتطفات عن برمجة الـ Pascal مثل method overloading ، الـ arrays الديناميكية ، والـ parameters الافتراضية .

## أدوات (الـ Command-Line):

يأتى Delphi بأدوات الـ command-line التى يمكنك استخدامها لإنشاء تطبيقات الـ Windows . ان النتائج هى نفسها التى تحصل عليها مع البيئة المتكاملة ، ولكن أدوات الـ command-line تسمح بالتطوير باستخدام محررين ثانويين بدلاً من بيئة Delphi . يمكنك أيضاً اختيار خيارات compiler لا يفهمها إلا الخاصة وإصدار ملفات خريطة رابطة للاستخدام اثناء إزالة الأخطاء . وتعتبر أدوات الـ command-line قيمة أيضاً فى إنشاء برامج اختبارية قصيرة ولتعليم تقنيات برمجة الـ Pascal من الكتب .

**على القرص المدمج:** ان الخطوة الأولى فى استخدام أدوات الـ command-line الخاصة بـ Delphi هى الوصول الى (command-line prompt) . يمكنك عندئذ إما ان تترك الـ Windows (استخدم أمر الـ Shut Down لزر الـ start واختر "Restart in MS-DOS mode" . أو ، اذا لم تكن تريد ان تترك الـ Windows ، افتح نافذة الـ DOS باستخدام الـ Start\Programs . أدخل path لتأكد مما اذا كان دليل الـ \bin الخاص بـ Delphi موجود على مسار النظام . اذا لم يكن موجوداً ، قم بتشغيل ملف المجموعة Delpath.bat فى القائمة (٢١-١) . يوجد هذا الملف فى دليل الـ Source\Misc على القرص المدمج . قد يكون عليك تعديل الـ string المسار اعتماداً على نسختك من Delphi واين ثبتها .



### القائمة (٢١-١) Misc\Delpath.bat

```
@echo off
rem
```

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

```
rem delpath.bat-Configure PATH for Delphi command line
rem
set path=%path%;"C:\Program Files\Borland\Delphi 4\bin"
echo Configured for Delphi command line
path
```

يمكنك الآن compile تطبيقات الـ CRT المستقلة، وإكمال مشروعات الـ Windows لـ Delphi. على سبيل المثال، مع وجود دليل الـ bin\ على المسار، استخدم نافذة الـ DOS للانتقال الى أى دليل به مشروع لـ Delphi. لتجربة هذا، إنتقل الى نسخة من تطبيق الـ Polyflow الخاص بهذا الكتاب. ومن الـ DOS prompt، أدخل أمراً مثل هذا:

```
cd "\Delphi 4 Bible\Source\polyflow"
```

لاحظ علامات التنصيص، والمطلوبة للدلالة ذات الكلمات المتعددة ذات الفراغات مثل Delphi 4 Bible لـ compile التطبيق، قم بتشغيل الـ dcc32 (compile) خط امر Delphi (ذى الـ ٣٢ بت). من الـ DOS prompt. هذا يؤدي الى تشغيل الـ Borland Pascal compiler. وبحث الـ compiler عن ملفات الـ pas. حسب النظام الافتراضى، لذلك، لكى تقوم بعملية الـ compile لتطبيق الـ Windows، يجب ان تدخل امتداد اسم الملف الـ .dpr. ادخل النص التالى بخط سميك. يجب ان تشبه شاشتك السطور الموضحة هنا، بالرغم من ان ارقام النسخ وتواريخ حق الطبع قد تختلف:

```
D:\Delphi 4 Bible\Source\Polyflow>dcc32 polyflow.dpr
Borland Delphi for Version 12.0 Copyright (c) 1983,98
Inprise Corporation
Polyflow.dpr(14)
```

15 lines, 0.75 seconds, 149128 bytes code, 4397 bytes data.

بفرض أنك لم تتلقى أى اخطاء compiler، يمكنك الآن كتابة Polyflow واضغط Enter لتشغيل البرنامج استخدم هذا الـ method لـ compile مشروعات والتي تريد ان تختارها مع خيارات بواسطة linker و command-line compiler للحصول على قائمة بالخيارات المتاحة، اكتب dcc32 واضغط Enter. على سبيل المثال، لـ compile برنامج وتعريف رمز شرطى، يمكنك إدخال أمر مثل هذا:

dcc32 -DDEBUGMODE polyflow.dpr

**تحذير:** إذا حاولت compile تطبيقاً قديماً تم كتابته باستخدام Delphi، قد تتلقى خطأ بأن ملف الـ resource ذي الـ ١٦ بت الخاص للمشروع لا يتماشى مع linker ذي الـ ٣٢ بت الخاص بـ Delphi. وقد يحدث هذا حتى بعد أن تقوم بتحديث و compile التطبيق بنجاح باستخدام البيئة المتكاملة لـ Delphi. والطريقة الوحيدة للخروج من هذه المشكلة هي أن تنشئ تطبيقاً جديداً ثم تضع ملفات الـ source القديمة في المشروع.



### تطبيقات الـ CRT:

إن النسخ الأولى من Delphi قد وفرت تطبيقات الـ CRT لكتابة فقط البرامج الاختبارية. وكان هذا ضرورياً لأن الـ Windows 3.1 ذي الـ ١٦ بت لم يكن يؤيد تطبيقات الـ console، كما يفعل الـ Windows 95 والـ Windows NT. إن قالب تطبيق الـ Crt لم يعد متاحاً، ولا في الـ WinCrt التي قدمت مخرجات تشبه الـ DOS في نافذة جرافيكية. ولكن، مع النسخ اللاحق لـ Delphi، يمكنك استخدام الخطوات التالية لإنشاء تطبيق CRT المستقل والذي يتم تشغيله من الـ DOS prompt.

إن تطبيق الـ CRT هو ببساطة برنامج نص فقط ولا يتطلب الـ from ولا يستخدم الـ components. ويعتبر تطبيق الـ CRT هو أسلوب الاقتصاد - وهو لا يكلفك الكثير من الوقت والمجهود، ولكنه يذهب بك حيث تشاء بسرعة.

### إنشاء تطبيق الـ CRT:

استخدام تطبيقات الـ CRT لاختيار الـ methods برمجة الـ Object Pascal، ولا اختبار الـ algorithms، ولتشغيل نماذج النص العادي من البرامج التعليمية للـ Pascal. لإنشاء تطبيق CRT، اقترح الخطوات التالية. قد يكون هناك طرق أخرى لتحقيق نفس النتائج النهائية، ولكنني أجد هذا الـ method هو الأسهل:

١- راجع الفصل السابق لتحديد الـ PATH الخاص بالنظام ليضم دليل الـ \bin الخاص بـ Delphi. بالرغم من أنه من الممكن compile تطبيقات الـ CRT

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

من داخل البيئة المتكاملة لـ Delphi، إلا ان compile تطبيقات الـ CRT المستقلة يعتبر اسهل من الـ DOS prompt. افتح نافذة الـ DOS prompt الآن وقم بتشغيل ملف مجموعة الـ Delpath المذكور فى الفصل السابق.

الملف موجود فى الدليل (source\misc)

٢- ارجع الى Delphi. اغلق كل الملفات الموجودة، اختر File\New.... اضغط الـ New page tab فى الـ New Items dialog، واختر الـ Text template. هذا يؤدى الى إنشاء ملف جديد، هو File1.txt، فى محرر الـ code.

٣- لإخبار Delphi ان الملف الجديد يحتوى على عبارات Pascal، احفظه فوراً مع امتداد اسم الملف pas. هذا يمكن من ابراز الـ syntax-highlighting، ويجعل الـ Code Insights متاحة (راجع الباب الأول).

٤- أدخل برنامجك واحفظه. يمكنك استخدام أى برنامج Pascal، ولكن اذا لم يكن لديك واحداً، إنسخ النص من القائمة (٢١-٢). يوجد هذا الملف على القرص المدمج فى دليل Source\Keytest.

٥- ارجع الى نافذة الـ DOS prompt. أدخل dcc32 -cc keytest. compile برنامجك- لا يجب عليك كتابة امتداد اسم الملف pas. ، ولكن ان تفعل فلا تضرر من هذا.

**تحذير:** عند compile تطبيقات الـ CRT المستقلة، لا تنسى ان تحدد

خيار الـ -cc. هذا يخبر الـ compile ان يقوم بتشغيل مخرجات

ومدخلات الـ command-line بدلاً من استخدام الـ Windows



. functions

القائمة (٢١-٢): Keytest\Keytest.pas. تطبيق CRT كمثال يمكنك استخدامه

لاختبار لوحة المفاتيح وتجربة ادوات الـ command-line الخاصة بـ Delphi.

اضغط Ctrl+C للإنتهاء

```
program KeyTest;
```

```
var
```

```
Ch: Char;
```

```
begin
  Writeln('Keyboard tester. Press +Enter. ');
  Writeln('To quit, press Ctrl+C, ');
  Writeln('or close the window. ');
  Writeln; { Output a blank line }
  repeat
    Read(Ch);    { Read from keyboard }
    Writeln(Ord(Ch):4); { Show its value }
  until False; { That is, "forever." }
end.
```

**ملاحظة:** من الممكن توليد console application باستخدام الـ Console Application check box. وهذه تعتبر أيضاً حيلة جيدة لتشغيل عبارات إزالة الأخطاء Writeln التي تعرض قيم أو تشير إلى عمليات برنامج متنوعة أثناء التطوير. ولكن تشير إلى عمليات برنامج متنوعة أثناء التطوير. ولكن من الأسهل غالباً compile وتشغيل تطبيقات CRT الخاصة من الـ prompt command-line.

إذا اكتشفت أن تشغيل تطبيق الـ Delphi Windows يفتح نافذة DOS، فإن هذا الخيار يكون في الغالب قد تم تشغيله. قم بإبطال اختياره، وأعد الـ compile لإصلاح المشكلة.

إذا كان لديك بالفعل Pascal source code، فيمكنك بالطبع استخدام محرر Delphi's code لاستعراض وتعديل عبارات البرنامج. استخدم الـ FileOpen... لفتح الملف (تأكد من أن الـ Files من النوع المحدد \*.pas). كواحد من امتدادات اسم الملف المعروفة). بعد حفظ تغييراتك استخدم الـ command-line compiler كما هو موضح لـ compile البرنامج، والذي يمكنك تشغيله من الـ DOS prompt. وإذا أردت، يمكنك استخدام محرر نص آخر لاستعراض، إنشاء، وتحرير ملفات الـ source code للـ CRT-يوجد العديد من الأنواع الجيدة في السوق.

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

استخدم ال Readln ، Read ، Writeln ، Write فى تطبيقات ال CRT على سبيل المثال ، عرف متغير string ، وجرب هذا البرنامج لتحفز المستخدمين على الإدخال ثم اعرض الاستجابة (لا يوجد هذا النص على القرص المدمج) :

```
program Yourname;
var
  S: String;
begin
  Write('Hi! What's your name? ');
  Readln(S);
  Writeln('Your name is ', S);
end.
```

**تحذير:** اذا تلقيت "Runtime error 103" عندما تحاول تشغيل تطبيق ال CRT السابق ، ففى الغالب قد نسيت ان compile مع خيار ال



-CC .

### function قوية:

بالرغم من ان ال Object Pascal Math unit الخاصة لـ Delphi توفر ال Power function التى يكون فيها قيمة الأس ، ويقدم هذا الفصل ال Power function التى تحدث exception لهذه العملية والعمليات الأخرى للقوة الغير المسموح به . لكى تستخدم هذه ال function فى ال code ، انسخ نوع ال exception class الثابت ، و function من القائمة الموجودة بعد الفصل الى أى module ، واستدع Power كما يوضح ال Test procedure لهذا البرنامج .

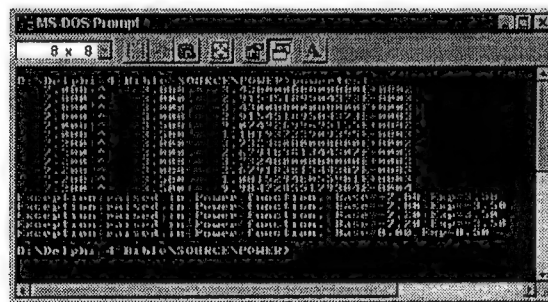
لقد نشرت نسخ من ال Power فى اماكن اخرى ، ولكننى ضمنت نسخة جديدة منها هنا لإظهار كيف تتمكن ال function الرياضية من استخدام معالجة ال exception لتعريف الاخطاء . على سبيل المثال ، من الغير مسموح به ان ترفع اساس سالبا الى اس كسرى أو ترفع الصفر الى أس غير صفرى اقل من واحد . ان



ال Power function هنا حدث EPower exception لهاتين الحالتين وغيرها من الحالات المشابهة. قم بتشغيل البرنامج لإظهار كيف يتعامل ال Test procedure مع هذه الانواع من الاخطاء.



On the  
CD-ROM



شكل (٢١-١): تشغيل برنامج الـ PowerTest يعرض هذه المخرجات في نافذة DOS prompt

القائمة (٣-٢١): Power\PowerTest.pas. هذا البرنامج يحتوى على الـ floating point، التي يمكن ان تكون الأس من نوع الـ

```
program PowerTest;
```

```
uses SysUtils;

const
  sFmt = 'Exception raised in Power function. Base=%f
        Exp=%f';

type
  { Declare exception class }
  EPower = class(EMathError)

  { Return Base raised to Exponent }
  function Power(Base, Exponent: Double): Double;
  function F(B, E: Double): Double;
  begin
    Result := Exp(E * Ln(B));
  end;
begin
  if Base = 0.0 then
    if Exponent = 0.0 then
      Result := 1.0
    else if Exponent < 1.0 then
      raise EPower.CreateFmt(sFmt, [Base, Exponent])
    else
      Result := 0.0
    else if Base > 0.0 then
      Result := F(Base, Exponent)
    else if Frac(Exponent) = 0.0 then
      if Odd(Trunc(Exponent)) then
        Result := -F(-Base, Exponent)
      else
        Result := F(-Base, Exponent )
      else raise EPower.CreateFmt(sFmt, [Base,
        Exponent]);
  end; { Power }
```

```

{ Test procedure }
procedure Test(Base, Exponent: Double);
begin
  try
    Writeln(Base:8:3, ' ^ ', Exponent:8:3, ' = ',
      Power(Base, Exponent));
  except
  on E: EPower do
    Writeln(E.Message);
  end;
end;

```

{ The following is the CRT application's main body. It merely calls the test procedure with various values. The final four values intentionally test the Power function's exceptions. }

```

begin
  test( 7, 3 );
  test( 7, -3 );
  test(-7, 3 );
  test(-7, -3 );
  test( 7, 3.5);
  test( 7, -3.5);
  test( 7.2, 3 );
  test( 7.2, -3 );
  test(-7.2, 3 );
  test(-7.2, -3 );
  test( 7.2, 3.5);
  test( 7.2, -3.5);
  { These four tests produce *expected* exceptions }
  test(-7, 3.5);
  test(-7, -3.5);
  test(-7.2, 3.5);
  test(-7.2, -3.5);
  test( 0, 0.5);
end.

```

## الباب الحادي والعشرين : تطوير مهارات Delphi الخاصة بك

ان ال Power function تحدث exception من ال EPower class ، المشتقة من ال EMathError class في ال SysUtils unit لـ Delphi . و ال EMathError class بدورها مشتقة من ال class الاساس Exception . (انظر الباب التاسع عشر لمعرفة مزيد من المعلومات حول البرمجة مع ال exception) .

لاستخدام ال Power function ، يجب ان توفر وصولاً الى ال exception classes الخاصة بـ Delphi . لكي تفعل هذا ، اضع SysUtils unit الى امر ال uses الخاص ببرنامجك (يستطيع Delphi ان يقوم بهذا بصورة تلقائية تدمج ال Power function في تطبيق ال Windows) . انسخ ال code الموجودة في القائمة (٢١-٥) برنامجك ، ناقص تعريف ال program وال uses ، وال end النهائية .

بالإضافة الى توفير function رياضية نافعة ، تظهر ال Power ايضاً طريقة جيدة لإنشاء exceptions رياضية توضح قيم خاطئة . على سبيل المثال ، لاجداث exceptions ، تنفذ ال Power هذه العبارة :

`raise EPower.CreateFmt(sFmt, [Base, Exponent])`

ان ال CreateFmt function ، الموروثة من ال Exception ، تأخذ string : arguments اوامر ، ومجموعة من القيم المحتجزة في اقواس مربعة . عندما يحدث exception ، يحتوى حقل ال Message الخاص بال EPower على string يوضح القيم التي سببت الخطأ . على سبيل المثال ، يعرض البرنامج الاختبارى رسالة خطأ لـ exception مثل :

Exception raised in Power function. Base=-7.20 Exp=3.50

## بعض ال Functions النافعة:

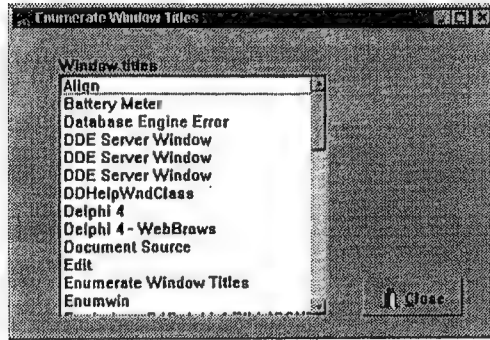
ان كل مبرمج لديه مكتبة من (subroutines) التي اثبتت قيمتها بمرور السنين . فيما يلي بعض الإضافات النافعة التي يمكنك ان تضعها بمكتبتك .

### ١- Callback functions :

ان ال callback function تعتبر (subroutines) في برنامجك يقوم ال Windows باستدعاءها . والاستخدام الامثل للـ callback function هو سرد

قائمة من العناصر مثل العناوين وال handles لكل النوافذ ، أو أسماء ال fonts المتاحة . والامثلة التالية توضح التقنيتين .

**على القرص المدمج:** توجد القائمة (٢١-٤) فى ملف Main.pas لمشروع ال EnumWin على القرص المدمج فى دليل ال Source\EnumWin . قم بتحميل ملف مشروع البرنامج فى Delphi ، ثم اضغط F9 لل compile والتشغيل . كما يوضح شكل (٢١-٢) ، يذكر البرنامج كل النوافذ النشطة حالياً (أى ، المعروفة لـ Windows) . وبالطبع ان عناوين النوافذ التى تراها تختلف غالباً عن التى تراها هنا . بالإضافة الى توضيح كيفية ايجاد عناوين النافذة ، يوضح البرنامج الطريقة الصحيحة للوصول الى كل النوافذ ، وهو ما قد تفعله لعدة اسباب - على سبيل المثال ، لتحديد ما اذا كان التطبيق الحالى فى حالة تشغيل بالفعل ، لإرسال رسالة الى كل النوافذ ، أو لتمكين المستخدمين من التحول الى نافذة معينة .



شكل (٢١-٢): برنامج ال EnumWin يستخدم ال callback function لذكر كل النوافذ "الحيلة"

القائمة (٢١-٤): EnumWin\Main.pas

unit Main;

interface

uses

Windows, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, StdCtrls, Buttons;

type

TMainForm = class(TForm)

ListBox1: TListBox;

Label1: TLabel;

BitBtn1: TBitBtn;

procedure FormCreate(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

(MainForm: TMainForm;

implementation

{ \$R \*.DFM }

{ Windows calls this function, passing in Handle a  
reference

to each alive window. Param is not used in this example. }

function EnumWinProc(Handle: HWND; Param: Longint):  
Boolean;

Stdcall; { Use this, not export, in Windows 95, 98 & NT }

var

Sz: array[0 .. 132] of Char; { Holds result of  
GetWindowText }

begin

Result := True; { Always successful }

{ Call Windows to obtain each window's caption, and  
then

add the returned string to the form's ListBox. }

if GetWindowText(Handle, Sz, Sizeof(Sz)) > 0 then

////////////////////////////////////

```
MainForm.ListBox1.Items.Add(StrPas(Sz));
end;
```

```
{ Enumerate all alive windows by passing to Windows the
  address of the preceding callback function. }
procedure TMainForm.FormCreate(Sender: TObject);
begin
  EnumWindows(@EnumWinProc, 0); { 0 is an unused
  parameter }
end;

end.
```

يستخدم البرنامج ListBox object واحد لحمل قائمة عناوين النوافذ. للملء الى ListBox object بهذه المعلومة، قمت بإنشاء ال OnCreate ل form التطبيق. ينفذ ال FormCreate procedure عبارة واحدة هي:

```
EnumWindows(@EnumWinProc, 0); { 0 is an unused
parameter }
```

ال EnumWindows يعتبر Windows procedure يتلقى arguments عنوان ال callback function، و parameter اختياري (أو Longint) - أي، object ذي ٣٢ بت على انه pointer. وهذا المثال لا يستخدم ال argument الثانية، لذا حددتها بصفر. وقد استخدمها برنامج آخر لتمرير العنوان الخاص بـ object البيانات الى ال callback function.

ان ال Callback function يجب ان يتم تعريفها بطريقة صحيحة وهذا أمر غاية في الأهمية لان ال compiler لا يتأكد من صحة syntax. عليك ان تستشير بحرص كل المصادر الممكنة للتوثيق لتحديد ال syntax الصحيح لل callback function التي تريد استدعاءه (ان ال Windows.pas في دليل ال Source\Rtl\Win الخاص بـ Delphi's يعتبر مفيد في هذه الحالة). وفي البرنامج، يتم تعريف ال callback function كما يلي:

```
function EnumWinProc(Handle: HWND; Param: Longint):
Boolean;
Stdcall;
```

**ملحوظة:** فى الـ Windows ١٦ ذى الـ ١٦ بت ، يتم تعريف هذه الـ function بطريقة مختلفة ، باستخدام الـ Export designator . فى الـ Windows 95 والـ Windows NT ذى الـ ٣٢ بت ، يجب تغيير الـ Export الى Stdcall . هذا يضمن تمرير الـ parameters فى ترتيبها الصحيح .

Note

ويتلقى الـ EnumWinProc إثنين من الـ parameters : الـ Handle الذى يشير الى نافذة واحدة ، والـ Param والذى ، كما ذكرت ، لا يتم استخدامه فى هذه الحالة . تقوم الـ function بإعادة نتيجة True أو False . يدخل الـ EnumWinProc النتيجة True للأستمرار فى تكرار كل النوافذ الحية حتى تنتهى ، ولكن يمكن ان تدخل False لإيقاف الاستمرارية .

تذكر أن الـ EnumWinProc ليست Windows function . إنها function فى برنامجك يستدعيها الـ Windows . ولهذا السبب ، ان الـ Windows يهتم فقط بمكان (عنوان) الـ function - فإسمها لا يهم . اذا اردت ان تعيد تسمية الـ EnumWinProc فهذا أمر يخصك . فالـ Windows لا يهتم . لقد ذكرت هذا لأنه فى العديد من القوائم الـ published ، تجد الـ Windows يستدعى function ذات هجاء مختلف .

ان الـ code الموجودة داخل الـ callback function تحدد الـ Result الخاص بالـ function بـ True ، ثم تستدعى الـ function GetWindowText الخاصة بالـ Windows باستخدام عبارة الـ if التالية :

```
if GetWindowText(Handle, Sz, Sizeof(Sz)) 0 then
```

```
MainForm.ListBox1.Items.Add(StrPas(Sz));
```

يقوم هذا الـ code بتمرير الـ GetWindowText الى الـ Handle الخاص بالنافذة و arguments إضافيتين : مكان لتخزين الـ caption الخاص بالنافذة ، وحجم هذا المتغير . وللتبسيط ، قمت بإنشاء Char array ذى ١٣٣ بايت ، وهو كبير بدرجة تكفى لحمل الـ caption ذو ١٣٢ حرفاً . هذا يعتبر كافياً لأى عنوان نافذة . ان الـ Sizeof function تقرر الحجم بالبايت للـ GetWindowText التى تم برمجتها على الـ تخزين المعلومات فى string object اكثر من المحدد لها .



إذا قامت الـ `GetWindowText` function بأرجاع `True`، إذن يوجد عنوان النافذة في المتغير `Sz`، والذي تم إضافته إلى الـ `ListBox1` الخاص بالـ `form`. لاحظ أن الـ `MainForm` قد تم الإشارة إليها لتعريف الـ `ListBox1 object`. وهذا يعتبر ضرورياً لأن الـ `callback function` ليست عضواً في الـ `TMainForm class` للتطبيق.

### : Procedure instances

أن الـ `Windows` ذي الـ ١٦ بت كان يتطلب مجهوداً خرافياً لإستدعاء الـ `procedures` والـ `functions` في `code` التطبيق الخاص بك. في الماضي، كان من الضروري إنشاء `procedure instance`، والمعروفة أيضاً بـ `thunk`، لاستخدام الـ `callback function` المستقلة. ولكن لم يعد هذا ضرورياً الآن مع الـ `Windows 95` و الـ `Windows 98` والـ `Windows NT`، والذي يستدعي الـ `functions` التطبيق مباشرة.

**على القرص المدمج:** لعرض كلتا التقنيتين، توضح القائمة (٢١-٥) الـ `source code` الرئيسية لمشروع الـ `Enumfon`. على القرص المدمج، يمكنك أن تجد ملفات البرنامج في دليل الـ `Source\Enumfon`. وبتشغيل البرنامج تحصل على قوائم بكل أسماء الـ `font` المتاحة- يمكنك استخدام برمجة مشابهة للمربع القائمة الذي يختار منه المستخدمون الـ `font`. لقد قمت بتحديث هذه التقنيات الجديدة، فبدلاً من إعادة إنشاء الـ `code`، قمت بإدخال `conditional compilation directives`. بهذه الطريقة، يستطيع البرنامج الـ `compilation` مع كل نسخ `Delphi` والـ `Windows`. إذا اردت تحديث تطبيقات قديمة، يمكنك استخدام تقنيات مشابهة للحفاظ على توافق الـ ١٦ بت يوضح شكل (٢١-٣) عرض البرنامج.



القائمة (٢١-٥): `Enumfon\Main.pas`. يوضح البرنامج كيفية سرد أسماء الـ `font`

باستخدام النسخ ذات الـ ١٦ والـ ٣٢ بت للـ `Delphi` والـ `Windows`

```
unit Main;
```

```
interface
```

```
uses
```

Windows, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, StdCtrls, Buttons;

```
type
  TMainForm = class(TForm)
    ListBox1: TListBox;
    Label1: TLabel;
    BitBtn1: TBitBtn;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

{ ===== }

{$IFDEF VER80} // Delphi 1.x, 16-bit Windows only

function EnumFontsProc(var LogFont: TLogFont;
  var TextMetric: TTextMetric; FontType: Integer;
  Data: TListBox): Integer; export;
begin
  with TStrings(Data) do
    Add(StrPas(LogFont.lfFaceName));
    Result := 1; { Continue enumeration until done }
end;

procedure TMainForm.FormCreate(Sender: TObject);
var
  Proc: TFarProc;
begin
  Proc := MakeProcInstance(@EnumFontsProc, HInstance);
```

```

    try
        EnumFonts(Canvas.Handle, nil, Proc, Pointer
(ListBox1.Items));
    finally
        FreeProcInstance(Proc);
    end;
end;

{ ===== }

{$ELSE} // Delphi 2 and higher, 32-bit Windows 95, 98, and NT
function EnumFontsProc(var LogFont: TLogFont;
var TextMetric: TTextMetric; FontType: Integer;
    Data: Pointer): Integer; stdcall;
var
    TheList: TStrings;
begin
    TheList := TStrings(Data); // Get StringList passed in Data
    TheList.Add(LogFont.lfFaceName); // Add font's name
    Result := 1; // Continue until done
end;

procedure TMainForm.FormCreate(Sender: TObject);
var
    DC: HDC;
begin
    DC := GetDC(0); // Get a device context handle
    try
        EnumFonts(DC, nil, @EnumFontsProc,
            Pointer(ListBox1.Items)); // Pass data to callback fn
    finally
        ReleaseDC(0, DC); // Release system resource
    end;
end;

```

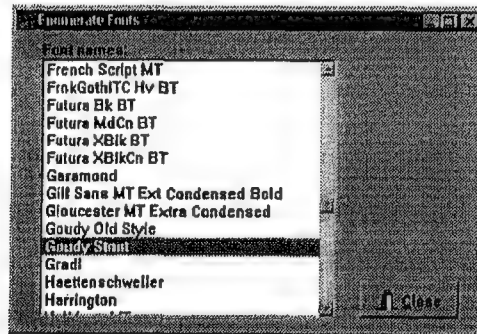
{ \$ENDIF }

{ ===== }

{\*  
The next procedure is \*far\* simpler than either of  
the preceding methods, and it also works with all versions  
of Delphi and Windows! Remember: Don't work harder  
than  
you have to; rather than call Windows subroutines and  
use  
callback functions, look for a Delphi component that  
does what you want. You can probably find one! }

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  Listbox1.Items := Screen.Fonts;
end;
*)

end.
```



شكل (٢١-٣): تطبيق ال Enumfon يذكر كل اسماء ال font المتاحة

### الحصول على معلومة اسم ال font:

قبل قراءة شرح ال FormCreate المستخدمة في ال Enumfon, ارجع  
ال ال procedure البسيط الموجودة في آخر القائمة داخل اقواس تعليق. ان ال  
FormCreate procedure يوضح ايسط method للحصول على معلومة اسم  
ال font قم بتعيين لـ Screen.Fonts ال object ال TStringList مثل خاصية ال  
Items في ListBox. وهذه العبارة:

**Listbox1.Items := Screen.Fonts;**

توضح واحداً من مبادئ البرمجة الـ Delphi استخدم دائماً objects و component methods اينما كان ممكن. لا تعقد الامور باستخدام callback functions واستخدام الـ Windows API subroutines والسبب الآخر لاستخدام objects هو ان الـ VCL palette الخاصة بـ Delphi تضم code توضح بعض غرائب الـ Windows ذي الـ ٣٢ بت. على سبيل المثال، ان الـ EnumFonts function القياسية للـ Windows لا تدخل دائماً كل اسماء الـ font مثل اليابان. وتتقلب الـ VCL TScreen palette على هذه المشكلة باستخدام الـ EnumFontFamilies لتتقديم قائمة اكثر شمولاً عما هي موجودة في الـ EnumFonts. ان الـ VCL الخاصة بـ Delphi تحتوي على العديد من الاشياء الأخرى التي يمكنك الانتفاع بها باستخدام objects بدلاً من استدعاء الـ Windows API functions.

توضح الـ Enumfon ثلاث تقنيات لذكر اسماء الـ font، إثنين منهم يستخدم الـ callback functions الخاصة بالـ Windows. والـ function الموضحة الأولى تكون لتطبيقات الـ ١٦ بت للـ Windows 3.1؛ والثانية لتطبيقات الـ ٣٢ بت للـ Windows 95, 98, & NT.

إذا كان يجب عليك استخدام الـ callback functions، فإليك طريقتين لتعقيد code برنامجك وبالطبع، هناك اسباب لإنشاء الـ callback functions مثل enumerating النوافذ واستخدام تقنيات الـ Windows الغريبة الغير متوفرة بواسطة الـ Delphi components. ولكن اريد ان اذكر ان التقنيات التالية يجب ان تحفظ لاوقات لا يوجد فيها بدائل اخرى.

في الـ Windows ذي الـ ١٦ بت، إنك تعرف الـ callback function مثل EnumFontsProc باستخدام الكلمة الأساسية الـ export. لاستدعاء الـ function، من الضروري أولاً ان تنشئ حالة الـ procedure instance تعمل موصل للاستدعاءات من الـ Windows عائدة الى code التطبيق. لكي تفعل هذا، يستدعي البرنامج الـ MakeProcInstance لـ function معينة، ثم يمرر النتيجة الى دالة الـ Windows التي تريدها. ها هو حد التمييز الاساسي:

**Proc := MakeProcInstance(@EnumFontsProc, HInstance);**

```
try
  EnumFonts(Canvas.Handle, nil, Proc,
    Pointer(ListBox1.Items));
finally
  FreeProcInstance(Proc);
end;
```

أولاً، يتم استدعاء الـ `MakeProcInstance` لإنشاء `procedure` instance من نوع الـ `TFarProc` للـ `callback function` `EnumFontsProc`، في هذا المثال. الـ `procedure instance` هذه يتم تمريرها الى الـ `EnumFonts` مع `device context handle`، و `pointer` للـ `TStringList`. وبالطبع، ان الـ `Windows` لا يعرف شيئاً عن الـ `TStringList` - الـ `parameter` الاخير، والذي يكون إختيارياً والذي يمرر إلى الـ `call back Function` الخاص بالبرنامج. من الضروري استخدام الـ `try-finally block`، كما هو موضح هنا، لـ `procedure instance` بحيث يتم تحرير بغض النظر عما اذا حدثت ايه `exceptions` اثناء استدعاء الـ `EnumFonts`. ولك ان تتوقع حدوث مشاكل ضخمة اذا لم تقم بمسح الـ `Windows resources` بهذه لطريقة.

ومن على السطح، تبدو تقنية الـ ٣٢ بت المقابلة بسيطة، ولكنها في الواقع تختلف عن الـ `method` السابق القديم. ان الـ `callback function`، وهي `EnumFontsProc`، يتم تعريفها بطريقة مشابهة، بالرغم من انني قد اوضحت طريقة افضل لتمرير الـ `Data parameter` على انه `Pointer`. بمعنى آخر، الـ `Data` تشير الى أى بيانات يتم تمريرها الى الـ `callback function`. ونتيجة الـ `function` تكون واحدة، وهي `Integer`. ولكن لاحظ ان الـ `sStdcall` يتم استخدامها بدلاً من الـ `export`. وهذا امر هام. تأكد من ان الـ `callback function` الخاصة بك تستخدم دائماً الـ `sStdcall`.

وداخل الـ `callback function`، يتم بدء متغير مؤقت، هو الـ `TheList`، باستخدام الـ `Data pointer` الذي تم تمرير على انه `parameter`. هذا لا يؤدي الى نسخ الـ `StringList`، انه يجعل الـ `TheList` يشير الى الـ `object` كمشير اكثر سهولة في العبارات اللاحقة. على سبيل المثال، لإضافة كل اسم `font`، يستدعي

البرنامج ال `TStringList.Add method` . لقد قدم ال `Windows` بالفعل اسم ال `font` فى ال `LogFont parameter` ، وهذا هو كل ما هو مطلوب لإضافة كل اسم الى ال `ListBox` الخاص بالبرنامج . وكما سبق ، ان الناتج لواحد يتم تمريره للخلف لاكمال العملية حتى يمرر ال `Windows` جميع ال `fonts` .

ان `OnCreate` الخاص بالبرنامج لل `form` الرئيسية يوضح كيف يستعد البرنامج ذو ال ٣٢ بت لاستخدام ال `callback function` . كان يمكن ان استعير ال `Canvas.Handle` ك `device context` ، ولكن بدلاً من ذلك ، اوضحت هنا طريقة اخرى للحصول على ال `object` اللازم . استدع ال `GetDC` ك `device context` جديدة ، وقم بتمريرها لل `EnumFonts` مع ال `nil` (parameter لا يستخدم) ، عنوان ال `callback function` الخاصة بنا ، و `pointer` الى ال `StringList` لحمل اسماء ال `font` .

مرة اخرى ، ان ال `try-finally block` مطلوب لضمان ان ال `device context` قد تم تحريرها بشكل سليم اذا ما تم حدوث أى `exceptions` فى استدعاء ال `EnumFonts` أو اثناء تنفيذ ال `callback function` . ولكن باستعادة ال `Canvas.Handle` ، يمكن تبسيط ال `code` الى الآتى . لقد كتبت النسخة الاطول لتوضيح الطريقة السليمة للتعامل مع ال `resources` النظام مثل `device context handles` . اذا لم تكن فى حاجة الى فعل هذا ، استخدم هذه النسخة الاقصر :

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  EnumFonts(Canvas.Handle, nil, @EnumFontsProc,
    Pointer(ListBox1.Items)); // Pass data to callback fn
end;
```

وللمراجعة ، ان الاختلافات الرئيسية بين تقنيات ال ١٦ بت وال ٣٢ بت هى :  
 \* يتم تعريف ال `callback function` ذات ال ١٦ بت مع الكلمة الاساسية `export` ؛ تستخدم النسخة ذات ال ٣٢ بت `stdcall` بدلاً من `export` .

ان ال `callback function` ذات ال ١٦ بت لا يمكن تمريرها مباشرة لل `Windows function` مثل ال `EnumFonts` . يجب أولاً ان تنشئ `procedure`

## الباب الحادي والعشرين : تطوير مهارات Delphi الخاصة بك

instanc باستدعاء ال MakeProcInstance، وتكرر هذه النتيجة الى ال callback function. يجب ايضاً ان تكون واثقاً من تحرير ال procedure instance.

\* ان ال Windows 95, 98, & NT يستطيع ان يستدعي ال functions التطبيق مباشرة، ولا حاجة لاستدعاء ال MakeProcInstance. قم بتمرير عنوان ال function كما تم التوضيح فى تطبيق ال Enumfon.

**فكرة:** مازال بإمكانك استدعاء ال MakeProcInstance وال FreeProcInstance فى تطبيقات ال ٣٢ بت؛ ولكن، ال Windows functions هذه تعود الآن دون تنفيذ أى code. ولتحويل ال ١٦ بت الى ٣٢ بت بسرعة، قد تستطيع ان تستبدل ال export بـ sStdcall وتعيد ال compile. لاحظ اننى قلت "قد". للحصول على افضل النتائج، قم بتحديث برامجك لتمرير عناوين ال callback function مباشرة كما هو موضح هنا.

يستخدم ال Enumfon اوامر ال conditional compilation directives لل compile المختلف اعتماداً على نسخة Delphi التى تستخدمها. ان النسخ المختلفة لـ Enumfon procedure تكون محتجزة داخل هذه الاوامر، والتى تعتبر نافعة فى إنشاء مشروعات تقوم بـ compile كل نسخ Delphi:

```
{IFDEF VER80} // Delphi 1.x
...
{$ELSE} // Delphi 2.x +
...
{$ENDIF}
```

## functions التاريخ والوقت:

ان ال System unit لل Object Pascal والتى تستخدمها كل التطبيقات تعرف نوع بيانات ال TDateTime على انه يعادل ال Float. وهناك مساعدة ال procedures وال functions التى تستخدم ال TDateTime لأى عملية تاريخ ووقت احتاجها.

ان جانب العدد الصحيح لتغير ال TDateTime يساوى عدد الايام التى مرت منذ ٣٠ ديسمبر، ١٨٩٩ (هذا يطابق تنسيق تاريخ ال OLE Variant-تواريخ Delphi 1.0 المحسوبة من عام ٢٠٠١). يخزن ال TDateTime الوقت على انه



fractional part للمتغير ، واليوم يبدأ من منتصف الليل الذى مضى . على سبيل المثال ، ٧٥ ، ٠ تمثل الساعة السادسة مساءً-ثلاثة ارباع اليوم بعد منتصف الليل .

استخدام الـ Trunc function للـ Pascal Object لاستخراج جانب اليوم من قيمة الـ TDateTime . واذا كان لديك إثنين من متغيرات الـ TDateTime وهما Date1 و Date2 ، فإن العبارة التالية تعين للعدد الصحيح N عدد الايام بين هذين التاريخين . ولان ما يهم هو القيمة المطلقة للطرح ، فلا يهم أى التاريخين تعين اولاً:

$$N := \text{abs}(\text{Trunc}(\text{Date1}) - \text{Trunc}(\text{Date2}));$$

وهناك function تاريخ ووقت اخرى وهى الـ IncMonth فى الـ SysUtils unit . وهذه الـ function تقوم بادخال قيمة TDateTime مساوية لتاريخ وهو عدد معين من الشهور بعد أو قبل تاريخ محدد . على سبيل المثال ، ان تمرير يوم الـ ٣١ من يوليو الى الـ IncMonth وطلب شهر واحد بعد هذا التاريخ يؤدى الى إدخال ٣٠ اغسطس . يمكنك استخدام هذه الـ function لبرمجة الاعمال الشهرية-طباعة تقرير فى آخر كل شهر ، مثلاً .

ان الـ IncMonth function ، والـ parameters الخاصة بها يتم تعريفها كما يلى :

```
function IncMonth(const Date: TDateTime;
  NumberOfMonths: Integer): TDateTime;
```

● **Date: TDateTime** : قم بتمرير أى تاريخ الى هذا الـ parameter ، لا يهم الوقت ، فقط التاريخ . (استخدم الـ StrToDate function لتحويل التواريخ فى تنسيق الـ string الى قيم TDateTime) .

● **NumberOfMonths** : قم بتمرير عدد الشهور التى يمكنك بواسطتها زيادة الـ Date . قم بتمرير قيمة سلبية لإنقاص الـ Date .

ان الـ IncMonth function تقوم بارجاع object TDateTime مساوياً لعدد من الشهور بعد أو قبل تاريخ محدد . على سبيل المثال ، قم بإضافة ListBox و Button على form ، اضغطوا مرتين Button object لإنشاء OnClick . اكمل

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

هذا ال procedure باستخدام ال code الموجودة فى القائمة (٢١-٦)، والموجودة على القرص المدمج فى دليل Source\Misc فى ملف Date1.pas. قم بتشغيل البرنامج واضغط الزر الملى ال ListBox بتواريخ من ٩٧/١/٣١ الى ٩٩/١/٣١. عند ما تفحص هذه القائمة، فستجد انها تشمل كل تواريخ نهاية الشهور مثل ٩٧/٢/٢٨ و ٩٨/١١/٣٠.

### القائمة (٢١-٦) MiscDate1.pas

```
procedure TForm1.Button1Click(Sender: TObject);
var
  D1, D2: TDateTime; // Date variables
  I: Integer; // for-loop variable
begin
  D1 := StrToDate('1/31/98');
  for I := -12 to 12 do
  begin
    D2 := IncMonth(D1, I);
    ListBox1.Items.Add(DateToStr(D2));
  end;
```

**ملحوظة:** ان ال IncMonth function قد تم برمجتها بشكل صحيح للتعامل مع التواريخ فى الالفية القادمة. على سبيل المثال، إنها تدخل بطريقة صحيحة التاريخ ٢٩/٢/٠٠ لان عام ٢٠٠٠ هو سنة عقدية نادرة، واستخدام ال IncMonth و functions التاريخ والوقت الأخرى ل Delphi يمكن ان يساعذك على تجنب ال "millenium bugs" أو اخطاء الألفية فى تطبيقاتك.

Note

### معلومات نوع وقت التشغيل:

يستطيع ال compiler ان يحدد من ال source code للبرنامج انواع ال objects التى يقوم بإنشاءها ولكن بالنسبة لل objects التى يتم انشاؤها فى وقت التشغيل، قد لا يكون ل source code متاحة، ويجب ان تحتوى ال objects على run-time type information (RTTI) أو معلومة نوع وقت التشغيل التى تصف ماهية هذه ال objects. وهذا موضوع شائك فى البرمجة بال

object-oriented . وتوجد حالياً مقاييس قليلة عن افضل format يمكن استخدامه لتعريف objects ووقت التشغيل . ان ال COM objects أى ال Common Object Model تقترب من هذه المقاييس ؛ ولكن ليست كل ال objects تتطابق مع ال COM .

وتوجد تعريفات ال RTTI فى ملف ال Typinfo.pas الموجودة فى دليل ال \Source\Vcl\... الخاص بـ Delphi . ولقد قامت ال Inprise Corporation بتخدير المبرمجين من ان تعريفات ال RTTI سوف تتغير فى إصدارات Delphi المستقبلية . وللتوصل الى تحديث بطريقة اسهل ، احرص على ان تعزل استخدامك عن هذه المعلومة .

**على القرص المدمج؛ وكمثال على كيفية استخدام ال Typinfo unit ،** فإن القرص المدمج يحتوى على Inidata.pas ، تم كتابته بواسطة المراجع الفنى لهذا الكتاب ، وهو Danny Thorpe . وتستخدم ال Danny unit ال RTTI object لكتابة قيم خاصية object فى ملفات البدء (ini) . للـ Windows . وباستخدام هذه ال unit ، يمكنك كتابة كل ما يخص ال dialog من check box ، نافذة تحرير ، radio button ، controls اخرى للـ unit . ، وتقرأ هذه القيم مرة اخرى بسهولة . وها هى بعض الملاحظات من Danny والتي تصف هذه ال unit :



" ان ال RTTI يمكن المطورين من كتابة code تعمل بـ objects بأسلوب عام لتشكيل الصفات التي يملكها ال object بدقة فى وقت التشغيل . ان ال Compiled code يتم الاشارة إليه على انها ال early bound لان كل قرارات الوصول والـ link تتم فى وقت ال compile . ويمكنك ال RTTI من إنشاء code تعتبر ال late bound ، وهذا يعنى ان الوصول والـ link يتم تحديدها فى وقت التشغيل . "

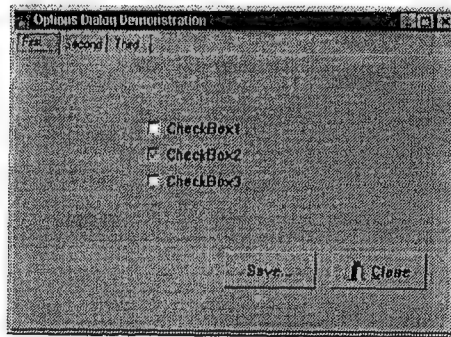
" ان ال IniData unit يمكن أن تكتب بيانات مستخدم متضمنة لأى form أو component . ان هدف ال unit هو تحميل وتخزين بيانات المستخدم فقط التي يحتويها ال component . ولقد تم بناء ال unit على بعض الاقتراضات التي ليس لها علاقة بالـ RTTI—مثل ان ال component الذي له خاصية واحدة هامة تحتوى على بيانات المستخدم ، وان هذه الخاصية بسيطة (ليست قائمة أو class ، مثلاً) . اذا اردت تخزين كل شئ فى form ، فقط اكتبها فى (stream) . "

## الباب الحادي والعشرين : تطوير مهارات Delphi الخاصة بك

**على القرص المدمج:** لتوضيح كيفية استخدام الـ Inidata.pas، قم بتحميل مشروع الـ Options في Delphi والذي يوجد على القرص المدمج في دليل Source\Options. قم بتشغيل البرنامج، واختر عبارته على صفحات الـ TabbedNotebook الثلاث للـ dialog يوضح شكل (٢١-٤) عرض البرنامج. اضغط زر الـ Save لحفظ اختيارات الـ Test.ini في دليل الـ C:\Windows. إنه البرنامج وأعد تشغيله لتحميل الخيارات من على القرص. ها هو ملف Test.ini قمت بإنشاءه:



```
[TMainForm]
CheckBox1=cbUnchecked
CheckBox2=cbGrayed
CheckBox3=cbChecked
RadioButton1=False
RadioButton2=False
RadioButton3=True
Edit1=Delphi 4 Bible
Edit2=IDG Books Worldwide, Inc.
```



شكل (٢١-٤): يوضح تطبيق الـ Options كيفية استخدام الـ Inidata unit لقراءة وكتابة خصائص الـ object في ملفات الـ ini.

**على القرص المدمج:** ان رأس قطاع ملف الـ ini. تساوى اسم الـ form class وكل عنصر يعتبر component object ذات اسم، وله قيمة تساوى خاصية component واحدة. (يمكنك حفظ خاصية object واحد). ان ملف الـ Main.pas الخاص بالمشروع في القائمة (٢١-٧)



يوضح كيفية استخدام ال Inidata unit . يوجد هذا الملف على القرص المدمج فى دليل Source\Options .

القائمة (٧-٢١) Options\Main.pas . هذا البرنامج يوضح كيفية استخدام ال

Source\Inidata unit ، والتي يتم تخزينها فى دليل ال Source\Inidata unit Main;

interface

uses

Windows, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, Inidata, StdCtrls, Buttons, TabNotBk,  
IniFiles, ComCtrls;

type

TMainForm = class(TForm)

TabbedNotebook1: TTabbedNotebook;

CheckBox1: TCheckBox;

CheckBox2: TCheckBox;

CheckBox3: TCheckBox;

RadioButton1: TRadioButton;

RadioButton2: TRadioButton;

RadioButton3: TRadioButton;

Edit1: TEdit;

Edit2: TEdit;

BitBtn1: TBitBtn;

BitBtn2: TBitBtn;

procedure FormCreate(Sender: TObject);

procedure BitBtn1Click(Sender: TObject);

private

procedure LoadOptions;

procedure SaveOptions;

public

{ Public declarations }

end;

var

MainForm: TMainForm;

implementation

{ \$R \*.DFM }

```
procedure TMainForm.LoadOptions;
var
  IniFile: TIniFile;
begin
  IniFile := TIniFile.Create('test.ini');
  try
    LoadDataFromINI(MainForm, IniFile);
  finally
    IniFile.Free;
  end;
end;
```

```
procedure TMainForm.SaveOptions;
var
  IniFile: TIniFile;
begin
  IniFile := TIniFile.Create('test.ini');
  try
    SaveDataToINI(MainForm, IniFile);
  finally
    IniFile.Free;
  end;
end;
```

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  RegisterINIDataProp('TCheckBox', 'State');
  RegisterINIDataProp('TRadioButton', 'Checked');
  RegisterINIDataProp('TEdit', 'Text');
  LoadOptions;
end;
```

```
procedure TMainForm.BitBtn1Click(Sender: TObject);
begin
  SaveOptions;
end;

end.
```

ان OnCreate للبرنامج يسجل ال component classes وخاصية واحدة للقراءة والكتابة فى ملف ال .ini ، على سبيل المثال ، لتسجيل TRadioButton ، ينفذ ال FormCreate هذه العبارة :

```
RegisterINIDataProp("TRadioButton", "Checked");
```

ولان ال TCheckBox components يمكن ان يكون لها ثلاث حالات ، يجب عليك ان تسجل خاصية ال states ، وليس ال Checked . يمكنك ايضاً تسجيل ال Edit باستخدام هذه العبارات :

```
RegisterINIDataProp("TRadioButton", "Checked");
```

```
RegisterINIDataProp("TEdit", "Text");
```

إختبر ال LoadOptions وال SaveOptions procedures كأمثلة على كيفية قراءة وكتابة خصائص ال object فى ملف ال .ini . تقوم ال LoadOptions بإنشاء object TIniFile ، وتستدعى ال LoadDataFromINI فى ال Inidata unit لقراءة الخصائص المحفوظة فى objects تابعة لل MainForm's . وبطريقة مشابهة تقوم ال SaveOptions باستدعاء ال SaveDataToINI لإنشاء أو تحديث ملف ال .ini .

## :Online Help

ان إنشاء ملف ال online help الخاصة بال Windows بيدك هو بالنسبة للتوثيق ل assembly language بالنسبة للبرمجة ان الخطوات صعبه ، والعملية تمنع الوقوع فى الخطأ ، والنتائج تجعلك تحن للايام التى لم يكن المستخدم يتوقع ان تكون البرمجيات على هذه الدرجة من التعاون .

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

قبل البدء فى شرح online help فإننى ادعوك للبحث عن مولد ملف ال help يمكن ان ينشئ ملفات ال help . الخاصة بال Windows بطريقة تلقائية ولا استطيع ان اشرح برنامجاً بعينه لأننى لم استخدمها كثيراً . ولكن، هذه البرامج توفر نظام ادخال لإدخال strings البحث، وقيم الفهارس، والكلمات الاساسية، والملاحظات، والعناوين، والنص .

**ملحوظة:** ان برمجيات اليوم تضم ملفات html. لتوفير online فى form لنمط ال World Wide Web page. فى الحقيقة، ان واجهة تطبيق ال Windows نفسها من المتوقع ان تنجذب ناحية ما يشبه واجهة تطبيق مستخدم ال Web بغض النظر عن حدوث هذا، فإن صعوبات توفير online لا تقل .

يأتى Delphi بنسخة من ال Help Workshop الخاصة بال Microsoft's، والتي يمكنك استخدامها لإنشاء ملفات online الخاصة ببرنامجك (فى ال .hlp format). يوجد البرنامج فى دليل ال Help\Tools الخاص به فى ملف ال Hcw.hlp. تصفح هذا الملف لتبدأ فى استخدام ال Help Workshop .

أى كانت الطريقة التى تنوى استخدامها، فبعد إنشاء ملف ال .hlp، قم بتعيين اسم الملف الخاص به لخاصية ال HelpFile الخاصة بال Application object. يجب ان تفعل هذا فى وقت التشغيل (فى ال OnCreate form مثلاً). أدخل عبارة مثل هذه:

Application.HelpFile := 'HelpMe.Hlp';

لتقديم مساعدة معينة ل objects فى form، قم بتعيين قيم ال help index. وهذه القيم اعداد صحيحة. على سبيل المثال، باستخدام ال Object Inspector، قم بتعيين ١٠٠٣ لخاصية ال HelpContext لل Button يمكنك تعيين قيم ال help context فى وقت التشغيل، ولكن عادة ما تقوم بهذا عندما تصمم ال form وعندما تضيف ال components الى النافذة.

**Tip** **فكرة:** وعدم الانتظار طويلاً لتعيين قيم ال help context هو الافضل غالباً. افعل هذا وانت تنشئ forms التطبيق، وحافظ على اثر قيم ال



context التى تستخدمها . قد تكون القيم سالبة . قد تريد اتباع method نظامى لترتيب ال indexes . على سبيل المثال ، قم بتعيين مدى محددة مثل ١٠٠٠ . . . . . ١٩٩٩ ل forms محددة .

ان قيم Help context لا يمكن بسهولة تحديثها ديناميكياً ، ويجب ان تحتفظ بحساب القيم التى تعينها . ومولدى ال help يقومون بصورة تلقائية بهذا ولكن عندما تقوم بإنشاء online يدوياً ، فإنك تحتاج (spreadsheet) أو قاعدة بيانات لتسجيل الموضوعات ، context-sensitive strings ، قيم context index ، والكلمات الاساسية .

افتح ملف ال help لموضوع معين ، استدع ال HelpContext method لل Application object . على سبيل المثال ، استخدم عبارة مثل هذه :

Application.HelpContext(Button1.HelpContext);

أو ، لكى تنتقل الى موضوع indexed بواسطة context-sensitive string ، استدع ال HelpJump كمايلى :

Application.HelpJump('Subject');

يقوم ال HelpJump بارجاع True أو False ، اعتماداً على ما اذا كان ال Application له ملف help معين . يمكنك عرض رسالة خطأ اذا ادخل ال HelpJump القيمة ب False :

if not Application.HelpJump('Subject') then

ShowMessage('Unable to open help file');

يجب ان يكون لتطبيقك قيمة HelpContext واحدة على الاقل مساوية ال Contents page index الخاصة بملف ال help . اذا لم تعين قيمة غير صفرية لخاصية ال HelpContext لل form الرئيسية على الأقل ، لا يستطيع المستخدم ضغط F1 لإظهار ال context-sensitive online .

**ملحوظة:** لان ال packages تقدم امكانية ان تكون ال dialogs وعناصر واجهة تطبيق المستخدم الأخرى المدرجة فى تطبيق ليس لها وصولاً لنظام ال help الخاص بالتطبيق ، فقد اضافت ال Inprise Corporation

Note

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

خاصية الـ HelpFile للـ TForm فى Delphi 3. عندما تكون نافذة الـ form نشطة ويطلب المستخدم الـ help بضغط F1، تلبى الـ VCL طلب الـ help باستخدام خاصية الـ HelpFile للـ form اذا كانت معينة، والا تستخدم الـ CVL خاصية الـ TApplication.HelpFile لهذا التطبيق.

### : Dynamic Link Libraries

ان الـ Dynamic Link Libraries، أو الـ DLL، تعتبر نوعاً من الـ subroutine package التى توجد فى ملف code منفصل، وتستطيع التطبيقات التشارك فيها. ان اغلب مكتبات الـ Windows تم توزيعها مع ملفات الـ DLL، ومكتبات الـ control المخصص يتم تقديمها عادة كـ DLL، ويمكنك تطوير انواع اخرى من المكتبات ذات الـ functions التى يمكن التشارك فيها فيما بين التطبيقات المتعددة.

باستخدام Delphi، يمكنك كتابة واستخدام الـ DLLs الخاصة بك، ويمكنك استدعاء functions فى الـ DLLs المطورة، فى نظم اخرى. يمكنك ان تضع component objects على forms وتضع النافذة فى DLL. والتطبيقات الاخرى، حتى تلك التى لم يتم تطويرها مع Delphi، يمكن ان تستخدم الـ DLL- انها package كامل فى مكتبة ملف code قابلة للتشارك.

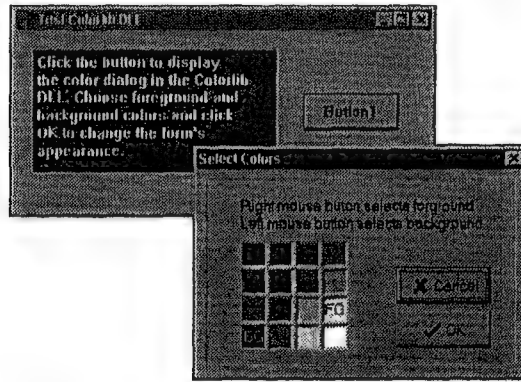
### إنشاء الـ DLL؛

كمشروع Delphi، تعتبر الـ DLL مكتبة لها procedures و functions يمكن ان تستدعيها التطبيقات الاخرى. فإن كل الاشياء الاخرى داخل الـ DLL مؤمنة وكأنها فى سيارة مصفحة. ولذلك، فإن احد المهام الرئيسية فى إنشاء DLL جديدة هو تعريف كيفية الوصول الى محتوياتها.

ان واحداً من الاسئلة المثارة هو عما اذا كانت الـ DLL تستطيع إنشاء form و component objects اخرى. والإجابة هى، نعم قطعاً! فى الواقع، ان هذا هو احد الاسباب الرئيسية فى كتابة Delphi DLL. يمكنك ان تضع form وأى Delphi components فى DLL. ان أى برنامج تم كتابته فى أى نظام تطوير Windows آخر يمكن ان يقوم بتحميل الـ DLL لتعرض الـ form. وهذه الطريقة

رائعة فى دمج امكانيات Delphi مع النظم الأخرى مثل الـ C ، والـ C++ ، والـ Visual Basic . (ولكن ، الـ ActiveX توفر طريقة أخرى) .

**على القرص المدمج:** عندما إنشاء DLL جديدة ، اختبار أولاً الـ form والـ components كتطبيق Delphi عام . قم بإنشاء وكتابة واختبار هذا المشروع كما تفعل مع أى مشروع آخر . وكمثال ، فإن تطبيق الـ ColorDll على القرص المدمج فى دليل الـ Source\Colordll ينشئ واجهة وخلفية dialog اللون الموضحة فى شكل (٢١-٥) . يوجد dialog باكملة على DLL منفصلة تستطيع التطبيقات الأخرى استخدامها . يوضح الـ objects الـ Delphi DLL والتي يمكن ان تكون لها form objects وأي component instances أخرى والتي يمكنك استخدامها فى التطبيقات .



شكل (٢١-٥): يعرض الـ Colorlib DLL اختياري  
اللون هذا ، باستخدام الـ ColorGrid object.

**على القرص المدمج:** كل الملفات التالية موجودة على القرص المدمج فى دليل الـ Source\Colordll . انسخ الدليل على محرك القرص الصلب الخاص بك ، ثم افتح مشروع الـ Colorlib فى Delphi . اضغط Ctrl+F9 للـ compile DLL . وهذا البرنامج غير قابل للتنفيذ ، ومحاولة تشغيله بضغط F9 تؤدي الى رسالة خطأ .



كتابة البرنامج ، قمت أولاً بتطوير dialog اللون على انه تطبيق Delphi عام . وعندما انتهيت ، قمت .

بتعديل ملفات الـ source code كما يلى يمكنك استخدام هذه الخطوات لتحويل أى مشروع Delphi الى DLL :

١ - اجعل اسم الـ form unit بـ ClrForm والمشروع بـ ColorLib ، احفظ المشروع كما تفعل فى المعتاد .

٢ - قم بتعريف procedure أو function فى قطاع واجهة التطبيق للـ form unit- وليس فى الـ form class . يقوم مستخدموا الـ DLL باستدعاء هذا (subroutine) للتمكن من الوصول الى المكتبة . يمكنك تعريف أى عدد من نظم الوصول كما تحتاجه الـ DLL الخاصة بك . على سبيل المثال ، ان الـ Clrform.pas تعرف الـ FBGetColors function كما يلى . وبما ان لغات اخرى يمكن ان تستدعى الـ function ، قم بتعريف انواع بيانات الـ Windows فقط (الـ WordBool مثلاً ، بدلاً من الـ Pascal's Boolean) . إتبع التعريف التالى بالكلمة الاساسية export ، التى تخبر الـ compiler ان يجعل هذه الـ function متاحة لمستخدمى الـ DLL :

```
function FBGetColors(var FColor, BColor: TColor):  
    WordBool; export;
```

٣ - قم بتنفيذ procedure أو function الوصول . توضح القائمة (٢١-٨) النتيجة النهائية لـ dialog اختيار اللون . سوف اشرح هذا الجزء من البرنامج بعد هذه الخطوات .

٤ - بعد ان تنتهى من كتابة الـ form unit ، قم بتحويل مشروع التطبيق الى DLL . لتفعل هذا ، اختر امر الـ View|Project Source لفتح ملف مشروع dpr . انتقل الى هذه الصفحة فى محور نص الـ unit ، وقم بالتغييرات التالية .

٥ - قم بتغيير الـ program الى library .

٦ - احذف Forms من امر الـ uses . (تحتاج المشروعات الى هذه الـ unit لاستدعاء الـ Application methods . والـ DLLs ليست تطبيقات ، وملفات مشروعاتها لا تحتاج الى الـ Forms unit .

٧- فيما بين امر ال Resource { \$R } والسطر الاخير من امر ال uses للمشروع ، اصف كلمة exports ، متبوعة باسم procedure أو function وعلى سبيل المثال FBGetColors unit . يمكنك إدخال واحدة أو أكثر من اسماء النظم الفرعية هنا .

٨- احذف كل العبارات فيما بين ال begin وال end ، تاركاً code block خالياً . توضح القائمة (٢١-٩) النتيجة النهائية لمشروع ال ColorLib .

٩- اضغط Ctrl+F9 لـ compile وإشاء ملف ال code dll . لا يمكنك تشغيل ال DLL ، لذا اذا ضغطت F9 ، فإنك تتلقى رسالة خطأ . ولكن هذا لا يمنع compile ال DLL ولا يسبب ضرراً .

#### القائمة (٢١-٨) : Colordll\Clrform.pas :

```
unit Clrform;

interface

uses
  Windows, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, ColorGrd, ExtCtrls;

type
  TColorForm = class(TForm)
    ColorGrid: TColorGrid;
    Label1: TLabel;
    Label2: TLabel;
    CancelBitBtn: TBitBtn;
    OkBitBtn: TBitBtn;
    Bevel1: TBevel;
    procedure CancelBitBtnClick(Sender: TObject);
    procedure OkBitBtnClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
    ColorForm: TColorForm;

function FBGetColors(var FColor, BColor: TColor):
WordBool;
    export;

implementation

{$R *.DFM}

{ Exit dialog via Cancel button }
procedure TColorForm.CancelBitBtnClick(Sender:
TObject);
begin
    ModalResult := mrCancel;
end;

{ Exit dialog via Ok button }
procedure TColorForm.OkBitBtnClick(Sender: TObject);
begin
    ModalResult := mrOk;
end;

{ Get foreground and background colors }
function FBGetColors(var FColor, BColor: TColor):
WordBool;
begin
    Result := False;
    ColorForm := TColorForm.Create(Application);
    try
        if ColorForm.ShowModal = mrOk then
            with ColorForm do
                begin
                    FColor := ColorGrid.ForegroundColor;
                    BColor := ColorGrid.BackgroundColor;
                    Result := True;
                end;
            end;
        finally
            ColorForm.Free;
        end;
    end;
```

=====

end;

end;

end;

القائمة (٩-٢١): Colordll\Colorlib.dpr. لتحويل مشروع تطبيق الى DLL، قم بتعديل ملف الـ dpr الخاص به كما هو موضح هنا  
library Colorlib;

{ Important note about DLL memory management:  
ShareMem must be

the first unit in your library's USES clause AND your project's (select View-Project Source) USES clause if your DLL exports any procedures or functions that pass strings as parameters or function results. This applies to all strings passed to and from your DLL-even those that are nested in records and classes. ShareMem is the interface unit to the BORLNDMM.DLL shared memory manager, which must be deployed along with your DLL. To avoid using BORLNDMM.DLL, pass string information using PChar or ShortString parameters. }

uses

// ShareMem, { Not required in this example }  
SysUtils,  
Classes,  
Clrform in 'CLRFORM.PAS' {ColorForm};

exports

FBGetColors;

{ \$R \*.RES }

begin  
end.

ان النسخ الحديثة من Delphi تقدم مشروع DLL يمكنك استخدامه لإنشاء DLLs جديدة من لاشئ. استخدم File|New...، اضغط ال New page tab، اختر ال DLL Block. بالرغم من ان اتباع الخطوات الموضحة هنا يتطلب مزيداً من العمل، الا اننى قمت بتطوير ال DLLs كتطبيقات مستقلة لأغراض اختيارية، ثم حولت ال code الى ال DLLs لمشاركتها فيما بين التطبيقات.

فى ال unit الخاصة بال DLL، وهى Clrform.pas، function ال FBColors تظهر كيفية قيام ال DLL بإنشاء ال form object. تقوم ال function بارجاع ال WordBool (أو نوع بيانات ال Boolean الخاص بال Windows) لتشير اذا ما كان المستخدم قد اغلق ال dialog بضغط (True) OK أو Cancel (False). أولاً، تبدأ ال function قيمتها المدخلة بـ False. بعد ذلك، تنشئ ال ColorForm object باستدعاء ال Create method لل TColorForm class's. ولأنك قد حولت التطبيق الى DLL، وهو لم يعد ينشئ form بصورة تلقائية، ويتعين على ال function ان تفعل هذا تحت تحكم البرنامج.

**فكرة:** استخدام دائماً ال try-finally block كما نوضح ال Clrform.pas لتضمن أنه حتى اذا حدث exception، يتم حذف ال form object.

Tip

وداخل ال try block، يستدعى البرنامج ال ShowModal method الخاص بال ColorForm's، والذي يدخل القيمة المعينة لل ModalResult بواسطة ال OnClick لل OK Button وال Cancel Button لل dialog's. اذا رجعت ال ShowModal بـ mrOk، يعين البرنامج اللون المتغيرة لل function. هذا يؤدي الى تمرير معلومة ال dialog خلفاً الى البرنامج وكعلامة على النجاح، يعين البرنامج True كنتيجة لك ال function.

**تحذير:** اذا قامت ال DLL الخاصة بك بتمرير أى string parameters أو نتائج لل function، أو اذا استخدمت arrays





ديناميكية، فيجب عليك استخدام الـ ShareMem unit فى الـ uses directive للتطبيق والـ DLL. ولا يحتاج البرنامج الاختبارى الموضح هنا الى هذه الـ unit، ولكننى أضفتها كتعليق لاحدد مكانها. وهذه القاعدة تنطبق حتى على الـ strings المحاطة فى سجل أو class object اذا قمت بتمرير بيانات strings بأى وسيلة من والى الـ DLL، اضف ShareMem لعبارة الـ uses كما هو موضح هنا. تذكر ان تفعل هذا فى كلاً من الـ DLL والتطبيق الذى يستخدمها.

يقوم الـ finally block باستدعاء الـ Free method الخاص بالـ ColorForm's لتدمير الـ form class وكل الـ components. هذا يعيد ذاكرة النظام الى حالتها فى بداية الـ function. يجب ان تكون بالغ الحرص فى التعامل مع الـ DLL عند إنشاء وتحرير الـ objects بطريقة سليمة.

ولضمان السلامة، يجب ان يتم السيطرة على الأخطاء فى الـ DLL من خلال الـ try-except block. هذا يمنع هروب أى exceptions من نطاق الـ DLL، وهو امر خطير لان معالجة الـ exceptions أمر غير قياسى خارج حدود التطبيق. فلا احد يستطيع ان يتوقع ما قد يحدث للـ exception إذا خرج من نطاق الـ DLL وراح يتجول فى الـ Windows. وإحدى الطرق لمعالجة الأخطاء الخطيرة هو ان يستدعى برنامج معالجة الـ exceptions الخاص بالـ DLL لتستدعى function مقدمة من البرنامج وتستخدم الـ DLL. والـ callback function هذه يمكن ان تحدث exception التطبيق. وهذا الحل يحافظ على الـ exception فى مداها-وتبقى الـ DLL exceptions فى الـ DLL؛ وتبقى exception التطبيق فى التطبيق.

### استخدام الـ DLL،

ان لديك الآن الـ Colorlib.dll فى دليل. بعد ذلك، يمكنك إنشاء تطبيق يستدعى function الخاصة بالـ DLL لعرض dialog اختيار لون. ولمجرد الهوى، يقوم التطبيق بتلوين واجهة الـ Memo الخاص به باستخدام قيم اللون المدخلة.

**على القرص المدمج:** يمكنك إنشاء التطبيق بنفسك أو تحميل مشروع الـ Testdll من نسخة لدليل الـ Colordll على القرص المدمج. توضح القائمة (٢١-١٠) الـ source code للـ Testdll.pas، والتي توضح



كيف يقوم تطبيق Delphi بتحميل واستخدام الـ DLL.

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

القائمة (٢١-١٠) : Colordll\Testdll.pas. هذا البرنامج يوضح كيفية تحميل واستخدام الـ Colorlib DLL من الفصل السابق

```
unit Main;
```

```
interface
```

```
uses
```

```
// ShareMem, { Not required by this example }  
Windows, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, StdCtrls;
```

```
type
```

```
TMainForm = class(TForm)  
    Button1: TButton;  
    Memo1: TMemo;  
    procedure Button1Click(Sender: TObject);  
private  
    { Private declarations }  
public  
    { Public declarations }  
end;
```

```
var
```

```
MainForm: TMainForm;
```

```
implementation
```

```
{ $R *.DFM }
```

```
{ Declare external function in the Colorlib.Dll }
```

```
function FBGetColors(var FColor, BColor: TColor):  
WordBool;
```

```
far;
```

```
external 'Colorlib';
```

```
{ Activate color dialog in the Colorlib.Dll }
procedure TMainForm.Button1Click(Sender: TObject);
var
  FColor, BColor: TColor;
begin
  FColor := Font.Color; { Form's text color }
  BColor := Color; { Window background color }
  if FBGetColors(FColor, BColor) then { Call DLL ! }
  begin
    Memo1.Color := BColor; { Assign window color }
    Memo1.Font.Color := FColor; { Assign text color }
  end;
end;

end.
```

فى التطبيق المضيف (التطبيق الذى يستخدم الـ DLL)، قم بتعريف نفس الـ function أو الـ procedure الذى تعرفه الـ DLL unit. ولكن، كما يوضح الـ Testdll.pas، استخدم form مختلفة بعض الشيء. قم بتعريف function. من الناحية الفنية، لا يجب عليك ان تفعل هذا اذا كنت واثقاً ان الـ code سيتم عمل compile لها باستخدام نسخة Delphi ذات ٣٢ بت، ولكننى ضمنت الكلمة الاساسية far على أية حال. كذلك بتعريف الـ function بـ external، يتبعها أسم الـ DLL string (ناقص امتداد الـ dll). هذا يخبر الـ compiler ان البرنامج سوف يقوم بإدخال الـ function.

والخطوة الاخيرة هى اسهل الخطى. استدع الـ function access كما يوضحOnClick للزر الخاص بالبرنامج. تفتح الـ DLL نافذة الـ dialog form، وتحكم تماماً فى كل الـ component objects. اذا رجعت الـ function بـ True، يستخدم البرنامج قيم اللون المختار والتى تم تمريرها الى الخلف عبر اثنين من الـ TColor parameters المتغيرة.

## Application Message Event Handlers والـ

تقدم الـ TApplication class عدداً من الـ events والتي يمكنك ان تكتب لها event handlers . ولان الـ Application object يكون غير مرئى فى الـ form ، والـ TApplication غير متاحة على الـ VCL palette ، فيجب عليك استخدام تقنية مختلفة لإنشاء الـ event handlers هذه . إتبع هذه الخطوات :

\* بتعريف الـ event handlers فى القطاع الـ public أو الـ private للـ form class (غالباً الـ Private) . لا تقم بتعريف الـ event handlers قبل الكلمة الاساسية private حيث يولد الـ Delphi التعريفات والـ component object والـ methods .

\* قم بتنفيذ الـ event handlers بكتابة الـ procedure . اجعل اسم الـ procedure مسبقاً باسم الـ form class . وبوجه عام ، استخدم نفس الـ format الذى يستخدمه Delphi عندما ينشئ الـ event handlers للـ objects التى تختارها باستخدام الـ Events page tab لنافذة الـ Object Inspector .

\* قم بتعيين الـ event handlers لخاصية الـ Application event : OnHint ، OnHelp ، OnException ، OnDeactivate ، OnActivate ، OnIdle ، أو OnMessage .

## Application event handlers للـ TApplication،

اتبع هذه الخطوات كمثال على تعيين للـ Application event handlers . يوضح البرنامج تقنية مفيدة للغاية يمكنك استخدامها لاداء عمليات خلفية دون استخدام الـ Timer :

١ - قم بإنشاء تطبيق جديد أضف Label و Memo على الـ form .

٢ - اصف التعريفين التاليين للقطاع الـ private للـ TForm1 class وتعتبر X هى عدد صحيح (والتي تم بدؤها بصفر عند إنشاء الـ form ويقوم AppOnIdle event handler بزيادته وعرضه فى الـ Caption الخاص بالـ Label . يجب ان يكون للـ event handler الـ parameters والـ form العامة الموضحة هنا ، ولكن يمكنك ان تسميه ماتشاء-ابحث فى events معينة عن الـ formats المطلوبة لـ handler procedures الخاصة بهم :

```
dx: Integer;
procedure AppOnIdle(Sender: TObject; var Done:
Boolean);
```

٣- قم بتنفيذ الـ AppOnIdle procedure بإضافة السطور التالية فى قطاع الـ implementation للـ unit. لاحظ ان اسم الـ form class يسبق اسم الـ procedure وتفصلها نقطة. يقوم الـ event handler بزيادة متغير الـ X الخاص وعرضه بتحويل الـ X الى string وتعيين النتيجة للـ Caption الخاص بالـ Label:

```
procedure TForm1.AppOnIdle(Sender: TObject; var Done:
Boolean);
begin
    Inc(X);
    Label1.Caption := IntToStr(X);
end;
```

٤- لربط الـ event handler مع الـ OnCreate event الخاص بالـ TApplication، قم بتعيين اسم الـ procedure الخاص بالـ event handler الى الـ event. يمكنك ان تفعل هذا فى أى مكان فى البرنامج، ولكن غالباً ما تقوم به فى الـ OnCreate form. اختر الـ form واضغط مرتين قيمة الـ OnCreate event الخاص بها لإنشاء هذا الـ event handler، وإدخل عبارة التعيين التالية:

```
Application.OnIdle := AppOnIdle;
```

٥- قم بتشغيل البرنامج، والذي يقوم بتحديث الـ Label باستمرار، حتى إنشاء ما تقوم بادخال نص فى نافذة الـ Memo.

**ملحوظة:** بالرغم من اننى استخدمت Label لهذا العرض، الا إن تغيير الـ Label caption يجعل نافذة الـ Label غير صالحة، مما يؤدى الى رسالة الـ wm\_Paint تنهى حالة الـ idle مبكراً. وهناك أنشطة اخرى قليلة الحدوث وتشمل استدعاء الـ MessageBeep، كتابة مخرجات للملف، أو ارسال رسالة الى تطبيق آخر.

## الباب الحادي والعشرين : تطوير مهارات Delphi الخاصة بك

يمكنك ان توقع برسالة Windows بانشاء event handler لل Application.OnMessage . هذا يساعدك على تحسين استجابات التطبيق على الرسائل المختلفة ، أو تقديم خدمات لرسائل غير معروفة . قم بتعريف OnMessage event handler كمايلي :

```
procedure AppOnMessage(var Msg: TMsg; var Handled: Boolean);
```

● **Msg**: هو سجل بيانات من نوع ال TMsg يحتوى على handle لنافذة ، وقيمة الرسالة ، و parameters ، و وقت ارسال الرسالة ، و احداثيات ال cursor . وهذه القيم تعتمد على الرسالة المحددة التي تم تلقيها .

● **Handled**: اذا كان OnMessage الخاص بك يتعامل مع رسالة ، حدد هذا ال parameter بـ True . اذا لم تكن تتعامل مع رسالة ، أو اذا اردت ان تستعيد اعماله الافتراضية ، حدد ال Handle بـ False .

**تحذير:** تمر عشرات الآلاف من الرسائل فى كل ثانية عبر ال TApplicationOnMessage . كن شديد الحذر فيما تفعله فى ال event message handler .



### Subclassing لرسائل ال Windows :

هناك طريقة اخرى للتعامل مع رسائل ال Windows وهى بكتابة method indexed لثابت رسالة معينة . على سبيل المثال ، يمكنك استخدام هذه التقنية للتقسيم الفرعى لاستجابة عنصر القائمة على اختبار أمر . تستطيع ال form البرنامج ان تنتظر رسالة الأمر ، تؤدي عمليات إضافية عندما يأتى الأمر على الخط ، وتكرر الرسالة لأداء نشاطها المعتاد .

جرب هذه الخطوات للتقسيم الفرعى لأمر ال File لقائمة Exit ، وعرض رسالة لها ازرار Yes و No يمكن للمستخدمين ضغطها لإنهاء البرنامج اختيار الأمر :

١ - ابدأ تطبيقاً جديداً . أضف ال MainMenu . اضغط ال object مرتين ، واستخدم ال Menu Designer الخاص بـ Delphi's لإنشاء قائمة File ذات أمر Exit .

٢- اختر أمر ال FileExit فى ال form وأضف Close فى ال Command event handler .

٣- قم بتشغيل البرنامج، واختر ال FileExit لأختبار ان هذا الامر يغلق نافذة البرنامج .

٤- إنك تقوم الآن بالتقسيم الفرعى لل event handler للأمر ال Exit بالامسك برسالة ال WM\_Command Windows الخاصة به ، والتي تم اصدارها عندما اخترت الأمر . أدخل تعريف ال message event handler التالى فى القطاع ال public أو ال protected لل TForm1 class . ان البرنامج له نفس اسم الرسالة ، ناقص ال underscore فى الثابت من الاسم . يتلقى ال procedure متغير ال TWMCommand الذى يحتوى على معلومات حول هذا النوع من رسالة ال Windows :

```
procedure WMCommand(var Message: TWMCommand);
message wm_Command;
```

٥- قم بتنفيذ ال WMCommand كمايلى . أدخل ال procedure فى قطاع ال implementation لل unit :

```
procedure TForm1.WMCommand(var Message: TWMCommand);
begin
  if Message.ItemId = Exit1.Command then
    begin
      if MessageDlg('Quit program?', mtConfirmation,
        [mbYes, mbNo], 0) = mrNo then
        begin
          Message.Result := 1; { Message was handled }
          Exit; { Exit without calling inherited }
        end;
    end;
  // inherited WMCommand(Message); { Old style }
  inherited; { Call inherited method }
  Message.Result := 1; { Indicate message was handled }
end;
```

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

٦- قم بتشغيل البرنامج، واختر الـ File|Exit. اذا أجبت بـ Yes ينتهى البرنامج، اذا اجبت بـ No، يستمر البرنامج. وهذا النوع من الاستفسار يمكن ان يقدم بطرق أخرى، ولكن اهمية التقنية الموضحة هنا هي ان الـ form class تعترض طريق رسالة الـ Windows لتقدم برمجة اضافية، أو لتحل محل اعمال الرسالة.

ان الـ event handler لرسالة الـ WMCommand يفحص حقل الـ ItemId فى سجل الـ Message. وهذا الحقل يساوى قيمة الأمر، والتي يتم تعيينها بصورة تلقائية لخصائص الـ TMenuItem.Command عندما تصمم Menu. استخدم خاصية الـ object لتجد هذه القيمة كما هو موضح هنا للـ tExit1.Command.

لعمل رسالة معترضة، استدع الـ inherited كما تم التوضيح هنا فى القسم السابق لـ Delphi، كان من الضروري ان تستدعى procedure موروثة بالاسم. وهذا لم يعد مطلوباً- استخدم فقط الكلمة الاساسية inherited كما فى المثال. ان استدعاء الـ inherited اختياري، ولكن ان لم تفعل هذا، يمكنك ان تمحو مستوى كامل من العمليات لرسالة معينة. اذا اردت ان تستبدل تماماً استجابة الـ code لرسالة معينة، إذن لا يجب عليك استدعاء الـ inherited. ولكن عادة يجب ان تفعل هذا:

**ملحوظة:** ان الـ event handler للـ Windows مثل الـ WMCommand يتم بتعريفهما الآن بشكل خاص الى الـ TForm-class مثلاً. والـ classes المشتقة لا تقوم بإستدعاء أو الإشارة الى التعريفات الخاصة للـ class السالفة. على المستوى الفنى، هذا هو السبب فى انك يجب ان تستخدم الـ inherited بلا argument كما كان مطلوباً فى الماضى.

حدد حقل الـ Result للـ Message parameter's بصفر لتشير الى ان الرسالة لم يتم التعامل معها. حدد الـ Result بواحد اذا تعاملت مع الرسالة. فى هذه الحالة، لا تهم قيمة الـ Result لان الـ procedure الموروث يحددها غالباً.

يمكنك استخدام التقنية الموضحة هنا للسيطرة على أى الـ event handler فى class. انظر التعريف الموجود فى ملف واجهة تطبيق الـ unit (Forms.pas)،



```

ShowMessage('Nonclient left button click'); //
Obsolete!
wm_NCRButtonDblClk:
ShowMessage('Nonclient right button click');
end;
inherited WndProc(Message);
end;

end.

```

قم بتنفيذ overridden window procedure باختبار الـ `Msg` فى متغير الـ `Message`. ان الـ `Msg` يساوى ثابت رسالة الـ `Windows`.

وغالباً، سوف تستخدم عبارة `case`، كما فى القائمة لتقارن الـ `Msg` مع واحدة أو أكثر من رسائل الـ `Windows` فى هذه الحالة، رسالتى الضغط المزدوج فى الـ `Client-area`. تذكر ان واحدة فقط منهما هى التى تعمل - ان الـ `wm_NCLButtonDblClk` مهجورة الآن، على الأقل لأغراض الـ `subclassing` للـ `event`. استدع الـ `WndProc` لعمل رسائل أخرى، أو يمكنك ان تحدد الـ `Message.Result` بقيمة غير صفيرية وتستدعى الـ `Exit` لتشير الى انك قد تعاملت مع الرسالة. ولكن يجب ان تمرر اغلب الرسائل على الـ `method` الموروث للعمل الافتراضى.

## الـ File Streams:

يشمل Delphi نظام `object-oriented file-streaming` والذى يمكنك استخدامه لقراءة وكتابة البيانات، أو الـ `components`، و `objects` أخرى فى ملفات القرص. توضح القائمة (٢١-١٢) كيفية استخدام الـ `TStream` classes والـ `TFileStream` لكتابة `Button object` على ملف قرص، ثم قراءته بعد ذلك. يوضح التطبيق أيضاً كيفية إنشاء `component object` تحت التحكم التام للبرنامج. قم بتشغيل البرنامج ثم اضغط `Create` لإنشاء زر وعرضه على الـ `form` اضغط `Write` لتسيير الزر الى ملف القرص المسمى `Test.stm` (يمكنك ان تجعل اسم الملف كما تشاء، ولا يجب ان يكون امتداد اسم الملف `.stm`). ان ضغط الـ `Write` يدمر الـ `button object` ويمحو من النافذة. اضغط `Read` لقراءة الـ `object` مرة أخرى من الـ `file stream`.

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

ان ال Main unit تحدد ال TButton object ، B ، بالسير من والى ملف قرص . انظر هذا التعريف فى قطاع ال unit implementation . للتحديد على الشاشة ما اذا كان الزر آتياً من برنامج أو من stream ، يعرض اثنان من BOnClick1 و BOnClick2 رسائل مختلفة باستخدام ال ShowMessage procedure . يقوم ال Procedure EnableButtons بتشغيل وابطال ازرار ال Create وال Write وال Read لاقتراح نظام صحيح من العمليات . وهذا على سبيل المثال يمنع إنشاء أو قراءة أكثر من Button object واحد فى كل مرة .

القائمة (٢١-١٢) : Compstrm\Main.pas

```
unit Main;
```

```
interface
```

```
uses
```

```
SysUtils, WinTypes, WinProcs, Messages, Classes,  
Graphics, Controls,
```

```
Forms, Dialogs, Buttons, StdCtrls;
```

```
type
```

```
TMainForm = class(TForm)
```

```
  CreateButton: TButton;
```

```
  WriteButton: TButton;
```

```
  ReadButton: TButton;
```

```
  CloseBitBtn: TBitBtn;
```

```
  procedure FormCreate(Sender: TObject);
```

```
  procedure CreateButtonClick(Sender: TObject);
```

```
  procedure WriteButtonClick(Sender: TObject);
```

```
  procedure ReadButtonClick(Sender: TObject);
```

```
  procedure FormDestroy(Sender: TObject);
```

```
private
```

```
{ Private declarations }
```

```
  procedure EnableButtons;
```

```
  procedure BOnClick1(Sender: TObject);
```

```

procedure BOnClick2(Sender: TObject);
public
{ Public declarations }
end;

var
MainForm: TMainForm;

implementation

{$R *.DFM}

const
fileName = 'Test.stm'; { The stream's filename }

var
B: TButton; { Object to be streamed to and from disk }

{ OnClick event handler before writing Button to stream }
procedure TMainForm.BOnClick1(Sender: TObject);
begin
    ShowMessage('Thanks. I needed that!');
end;

{ OnClick event handler after reading Button from stream }
procedure TMainForm.BOnClick2(Sender: TObject);
begin
    ShowMessage('Hello. It's nice to be back!');
end;

{ Enable and disable buttons depending on whether the
Button
object and Test.Strm file exist }
procedure TMainForm.EnableButtons;

```

```

begin
  CreateButton.Enabled := False; { Disable all buttons }
  WriteButton.Enabled := False;
  ReadButton.Enabled := False;
  if B nil then
  WriteButton.Enabled := True { Write if B exists }
    else begin
      CreateButton.Enabled := True; { Create if B doesn't
        exist}
      if FileExists(fileName) then
        ReadButton.Enabled := True { Allow read if file
          exists }
        end;
    end;
end;

{ Initialize other buttons to control clicking order }
procedure TMainForm.FormCreate(Sender: TObject);
begin
  EnableButtons;
end;

{ Create Button object entirely under program control }
procedure TMainForm.CreateButtonClick(Sender:
TObject);
begin
  B := TButton.Create(Self); { Create the Button object }
  B.Parent := Self; { Assign parent object }
  B.Left := 240; { Assign left position }
  B.Top := 72; { Assign top position }
  B.Caption := 'Click me!'; { Assign label }
  B.OnClick := BOnClick1; { Assign first event handler }
  EnableButtons; { Update operation order }
end;

```

```

{ Write Button object to file stream, and then destroy it }
procedure TMainForm.WriteButtonClick(Sender: TObject);
var
  Stream: TStream;
begin
  { Create a new disk file stream }
  Stream := TFileStream.Create(fileName, fmCreate);
  try
    Stream.WriteComponent(B); { Write Button object }
    B.Free;                  { Destroy Button object }
    B := nil; { Prevent form's OnDestroy from Freeing
again }
    EnableButtons;          { Update operation order }
  finally
    Stream.Free; { Destroy the file stream object }
  end;
end;

```

```

{ Read Button object from stream }
procedure TMainForm.ReadButtonClick(Sender: TObject);
var
  Stream: TStream;
begin
  { Create stream object in read-only mode }
  Stream := TFileStream.Create(fileName, fmOpenRead);
  try
    B := TButton(Stream.ReadComponent(nil)); { Read &
Create B }
    B.Parent := Self; { Always assign parent! }
    B.OnClick := BOnClick2; { Assign second event
handler }
    EnableButtons; { Update operation order }
  finally

```

```
Stream.Free; { Destroy the file stream object }
end;

end;

{ Destroy button if user closes app before completing test }
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  if B nil then B.Free;
end;

{ Register the TButton class with Delphi stream system }
begin
end.
```

ان CreateButtonClick ينشئ button object جديد باستدعاء الـ Create constructor للـ TButton class وتحدد ناتج الـ object Reference لـ B. لكى تجعل الـ object مراثياً، فيجب ان يكون له Parent، وهو فى هذه الحالة الـ form الرئيسية. ان اغلب الـ objects تحتاج الى قيم Left و Top لتحديد موضعها- يمكنك ايضاً تحديد الـ Width والـ Height للـ object، ولكن هذه الخصائص والخصائص الاخرى لها قيم افتراضية. وحتى يفعل الزر شيئاً عندما تضغطه، يعين الـ BOnClick1 procedure للـ Button OnClick.

ان الـ WriteButtonClick يوضح كيفية إنشاء file stream جديد. افعل هذا باستدعاء الـ Create constructor لـ TFileStream مع arguments، اسم ملف وثابت نمط مثل fmCreate، واحفظ الـ object Reference الناتج فى متغير TStream أو TFileStream.

جدول (٢١-١): يعرض الثوابت التى يمكن تمريرها إلى TFileStream

الأثر	Constant
يفتح ملف للقراءة	O fmOpenRead
يفتح ملف للكتابة	fmOpenWrite
يفتح ملف للقراءة والكتابة	fmOpenReadWrite

يفتح ملف للاستخدام المختصر للتطبيق	fmShareExclusive
يمنع تطبيقات أخرى من الكتابة على الملف	fmShareDenyWrite
يمنع تطبيقات أخرى من القراءة من الملف	fmShareDenyRead
يسمح لتطبيقات أخرى من القراءة والكتابة على الملف	fmShareDenyNone
ينشئ ملف جديد ويفتحه للكتابة	fmCreate

استخدم تعبيرات ال OK المنطقية لدمج ثوابت نمط الملف . على سبيل المثال ،  
يفتح التعبير التالي ملفاً للقراءة والكتابة ، ولكن يمنع التطبيقات الأخرى من الكتابة  
على الملف مادام مفتوحاً :

fmOpenReadWrite or fmOpenDenyWrite

استدع ال WriteComponent method الخاص بال TStream وكتابة  
object على ال stream . يمكنك كتابة أى عدد تحتاجه من ال components ،  
ويمكن ان يكون من نفس النوع أو انواع مختلفة . بعد كتابة ال Button ، يقوم  
التطبيق بهدمه ويحدد B ، ب nil . اذا اغلقت التطبيق ، فإن هذا يمنع ال  
form OnDestroy من تحرير نفس ال object مرة أخرى ، مما قد يسبب General  
Protection Fault (GPF) .

ان ال WriteButtonClick يستخدم ال try-finally block لضمان ان ال  
Stream قد تم تحريره حتى اذا حدث exception . استخدم دائماً ال try-finally  
بهذه الطريقة فى ال procedures وال functions الخاصة بك التى تنشئ ال  
TFileStream object .

ان ال ReadButtonClick Procedure يقرأ ال streamed object من  
على القرص . ومرة أخرى ، تستدعى الخطوة الأولى ال Create constructor  
الخاص بال TFileStream ولكن هذه المرة باستخدام نمط ال fmOpenRead .  
وهذه العبارة تقرأ ال object على ال file stream وتعينه الى B :

B := TButton(Stream.ReadComponent(nil));

إستدعى ال TStream ReadComponent method لقراءة component أو file stream . قم بتمرير مرجع الى ال ReadComponent component لقراءة object فى object موجودة بالفعل ، أو تمريره كما فى هذا المثال لكى تجعل ال ReadComponent ينشئ object جديد تماماً ويرجع ال ReadComponent نوع ال TComponent ، لذا يجب عليك استخدام type-cast لتعيين النتيجة الى نوع معين من مرجع ال component مثل B .

بعد قراءة object من على ال stream ، تأكد من ان تجعل له Parent- فهذه المعلومة ليست محفوظة فى ال stream لانه form مختلفة أو نفس ال form فى موضع مختلف فى الذاكرة قد تملك ال object . يجب عليك ايضاً أن تعيد تعيين أى event handlers وكقاعدة عامة ، أى معلومة عن عنوان معين (event handlers ، مشير لـ object اخرى ، وخاصية ال Parent) قد لا تكون على ال stream لان هذه المعلومة تكون مختلفة بعد قراءة ال objects .

قبل ان يستطيع التطبيق قراءة وكتابة ال object فى file stream ، يجب عليه ان يسجل objects classes مع أسلوب ال file streams الخاص بـ Delphi . يمكنك ان تفعل هذا فى ال OnCreate form ، ولكن لضمان ان العمل قد تم باسرع ما يمكن ، اصف الكلمة الاساسية initialization قبل عبارة ال end فى ال unit . وفيما بينهما ، أضف استدعاءات لل RegisterClass لكل class تكون هذه ال object Register classes module أو stream objects .

## تطبيقات ال Internet:

ان طبعة ال Client/Server من Delphi تأتى بمجموعة كاملة من ال components لتطبيق تطبيقات ال Internet وال Intranet . (ان الإنترنت هى شبكة تشبه ال Internet فى ال protocols وتنظيمها ، ولكنها مقصورة على المجموعة المحلية من الحاسبات-مثلاً ، كل النظم فى شركة ما) . وهذه ال components تعتبر نافعة ، ليس فقط للوصول الى مواقع ال Internet وال Intranet ، ولكن ايضاً لتطوير برمجيات قاعدة البيانات (Remote-access) . على سبيل المثال ، Inprise Corporation uses Inprise components فى Delphi الاخبار عن الأخطاء فى النسخ والذي اصبح متاحاً لمختبرى البرمجيات .



## Internet components:

يوفر Delphi مجموعة من ال components لبرمجة ال Internet و al intranet. نسخة ال Client/Server تقدم components إضافية كما هو موضح فى شكل (٢١-٦). ولان قليل من قراء هذا الكتاب سوف يحصلون على نسخة ال Client/Server الباهظة الثمن، فإننى اقدم هنا ال components المقدمة مع ال Delphi Professional فقط.



شكل (٢١-٦): ال Internet components لطباعة ال Client/Server من Delphi

**ملحوظة:** ان Inprise Corporation Internet component قد أجرى لها بعض التغييرات منذ الاصدار الأول. ان ال components FTP، NNTP و components الأخرى المؤسسة على ال -ActiveX قد تم استبدالها فى Delphi 4 بال VCL components الأصلية المسارية لها والتي تم إنشاؤها بواسطة NetMasters. وهذا التغيير لا يجب ان يؤثر على اسماء ال classes المستخدمة فى النسخ الأولى، ولكن؛ ولكن نسخة قبل الاصدار الخاصة بى من Delphi لم تكن تشمل كل ال classes، بالرغم من ان ال Inprise Corporation اكدوا الى ان الاصدار الاخير من ال Internet components classes سيكون مطابقاً للنسخ الاولى.

ان ال Internet components الخاصة بـ Delphi ليس لها صور بصرية وقت التشغيل، وهى موضحة كايقونات فى نافذة ال form وتعتبر كل ال -ActiveX controls محاطة باغلفة Delphi component. وبحسب ترتيب ظهورها على الشاشة، فإن ال Internet ActiveX components من Delphi هى:

### • (FTP (Internet File Transfer Protocol): ينقل الملفات

والبيانات بين الحاسوب المحلى والبعيد.

### • HTML (Hypertext Markup Language): يقدم مستعرض وثيقة

HTML. ويمكن ايضاً ان يستخدم للـ parsing، وليس لعرض، وأمر وثيقة ال HTML.

● **HTTP (Hypertext Transport Protocol)** : يسترجع وثائق ال HTTP دون تصفح أو إمكانيات عمل الصورة.

● **NNTP (Network News Transfer Protocol)** : يقدم لمجموعات الاخبار قراءة وارسال الاخبار.

● **POP (Post Office Protocol)** : يحصل على البريد من ال UNIX أو انواع اخرى من ال servers التي تؤيد ال POP3.

● **SMTP (Simple Mail Transfer Protocol)** : يتوصل الى ال SMTP mail servers ويقدم ارسال البريد.

● **TCP (Transmission Control Protocol)** : لتطبيقات ال Client/Servr التي تحتاج خدمات شبكة ال TCP.

● **UDP (User Datagram Protocol)** : ينفذ ال WinSock لتطبيقات ال Client/Servr. يمكن ايضاً ان يرسل ويستقبل بيانات ال UDP.

### تطبيقات ال Internet الخاصة بـ Delphi:

يقدم Delphi العديد من التطبيقات التي توضح كيفية استخدام ال Internet components . وكمقدمة سريعة من برمجة ال Internet ، جرب متصفح ال Web الموجود في دليل ال Demos\Internet\HTML لـ Delphi's .

يوضح هذا البرنامج كيفية إنشاء متصفح Web كامل باستخدام ال HTML component واحد فقط . قم بتحميل ملف مشروع ال Webbrowser.dpr على هذا الدليل في Delphi's ، اضغط F9 للـ compile والتشغيل يوضح شكل (٧-٢١) نافذة البرنامج بعد اتصالها لموقع ال Web الخاص بالـ Borland's .

لطلب وثيقة ال HTML على ال Internet ، قم بإضافة ال HTML component على form ، وقم ببرمجة ال event handler لاستدعاء ال RequestDoc method للـ component كمايلي :

```
HTML1.RequestDoc(URLs.text);
```



شكل (٢١-٧)، يوضح ال Web الخاص ب Delphi's برمجة  
ال Internet مع ال components

يفترض أن ال URLs هي ComboBox component أو component آخر له خاصية نص محددة بال URL الذي تريده . يمكنك أيضاً تمرير string حرفي لل RequestDoc :

```
HTML1.RequestDoc('www.inprise.com');
```

يعتبر ال HTML component object مقيداً فى الحصول على نص المصدر لوثيقة ال HTML . لكى تفعل هذا ، اتصل بالموقع باستدعاء ال RequestDoc ، ثم قم بالوصول الى خاصية ال SourceText لوثيقة ال HTML . وهذا هو قائمة string تحتوى على نص من الوثيقة .

ان متصفح ال Web الخاصة ب Delphi يستخدم خاصية ال SourceText لعرض مصدر ال HTML فى نافذة ، مضيفاً النص الى Memo يستخدم هذه العبارات :

```
Memo1.Lines.Clear;  
Memo1.Lines.Add(HTML1.SourceText);
```

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

وهناك اثنين من ال HTML events تعتبر مقيدة فى استكشاف بداية وتوقف استرجاع الوثيقة. اختر HTML object واضغط باب ال Events على ال Object Inspector. ثم اضغط مرتين ال OnBeginRetrieval وال OnEndRetrieval لإنشاء events handlers لكل منهما وهما :

● **OnBeginRetrieval** : يتم استدعاؤه عند بداية تحميل الوثيقة .

● **OnEndRetrieval** : يتم استدعاؤه عند إنهاء تحميل الوثيقة .

ويستخدم متصفح ال Web هذين ال events لتشغيل وإبطال زر ال Cancel للبرنامج . (من الواضح ان عملية ال Cancel غير لازمة بعد ان يتم تحميل الوثيقة كاملة) . يمكنك ان تفكر فى استخدامات اخرى مفيدة لهذين ال events- عرض رسالة على الشاشة تخبر المستخدمين ان الوثيقة يتم تحميلها ، مثلاً .

وبالطبع يمكنك استخدام العديد من ال Internet components ، العديد من ال events والخصائص الاخرى لتطوير تطبيقات ال Internet .

### مزيداً من الافكار :

إننى اجد ان الافكار التالية ، التى تناسب فى ملحوظات مع بعض ملفات مجموعة ال DOS ، نافعة للغاية اثناء استخدام وتعلم Delphi . وقد تجدها كذلك انت ايضاً .

### إنقاص حجم ملف ال code :

فيما يلى إثنين من التقنيات الهامة التى تستطيع ان تقلل حجم ملف ال code بدرجة كبيرة :

\* إغلاق معلومة ازالة الاخطاء ، التى يتم كتابتها على ملف ال exe. اثناء debugging sessions. اختر ProjectOptions... واضغط ال Linker page tab. أغلق ال TD32 debug info check box. هذا يمنع ال compiler من نقل معلومات ازالة الخطأ الى ملف ال code.

\* استخدم packages وقت التشغيل . يقوم Delphi بإنشاء ملف exe. كامل يحتوى على code تطبيقك زائد كل ال components code على ال VCL.

هذا يجعل ملف الـ code ملائماً للتوزيع لأنه يجب عليك تقديم ملف واحد فقط للمستخدم النهائي . ولكن النسخ المتعددة من ملف code التطبيق تحتوي على نسخ لا داعى لها من الـ VCL . قم باستئصال هذه النسخ الزائدة باختيار ProjectOptions... ، اضغط على Packages page tab ، وقم بفتح الـ check box الذى يحمل بطاقة Build مع packages وقت التشغيل . وهذا الاختيار البسيط يمكن ان يقلل حجم الـ code الخاص بك بـ ٥٣٢ ك أو أكثر . ولكن ، لان الـ packages يجب ان تكون متاحة لتشغيل البرنامج ، فأى محفوظات لمساحة القرص يتم تحقيقها فقط اذا كان لديك تطبيقات Delphi متعددة على نفس النظام .

لكلا الـ methods ، اعد بناء تطبيقك لإنفاص ملف exe . التام . من المهم ان تعيد البناء بحيث يتم إعادة compile كل الـ methods لا تضغط فقط F9 للـ compile والتشغيل للتطبيق كما تفعل عادة . لكى تنشئ ملف exe . التام ، اختر ProjectBuild .

ان packages وقت التشغيل ايضاً تقلل استخدام الذاكرة . لان الـ packages ليس أكثر من مجرد compiled DLL بشكل خاص ، فإن الـ code تكون مشتركة بين كل تطبيقات Delphi فى حالة تشغيل .

**ملحوظة:** عند استخدام الـ packages ، التى يجب ان توفرها للمستخدمين النهائيين - على سبيل المثال ، ملف VCL40.bpl فهذا الملف وملفات الـ bpl الاخرى المطلوبة يجب ان يتم نسخها على دليل الـ C:\Windows\System الخاص بالمستخدم النهائي .

Note

### : Focus changes

فى بعض نسخ Delphi ، لا نوصى بأداء أى اعمال تؤدي الى focus change فى الـ event handler الذى يكون هو نفسه جزء فى الـ focus change . على سبيل المثال ، ان محاولة عرض الـ message dialog فى الـ Edit OnExit ، يمكن ان يجعل البرنامج يفقد القدرة على التحكم فى controls اخرى .

وهذه المشكلة لم تعد موجودة فى Delphi 3 والنسخة التالية . لتعرف اذا كان نظامك يعرض هذا السلوك ، ضع بعض الـ Edit objects على form واستدع الـ

OnExit ShowMessage ل object واحد. قم بتشغيل البرنامج وانتقل من controls الى controls بعد ان تغلق ال message dialog، قد يكون لديك مشكلة فى نقل ال forms الى controls اخرى- باعطاء ال dialog يعطى ال control التالى ال focus، وهو ما تفعله ال OnExit عندما تعرضه. على سبيل المثال، ان ضغط ال Tab بعد اغلاق ال dialog لم يحرك ال forms الى نافذة ال Edit التالية كما هو متوقع. وهذه المشكلة ليس خطيرة.

### : Multiple program instances

كما تعرف، يمكنك تشغيل instances متعددة من لبعض برامج Windows، ولكن بعضها يقيّدك بان تقوم بتشغيلها مرة واحدة فقط. ويشترط ال Microsoft ان ال instances الجديدة لتطبيق ما يتم إنشائها عندما يقوم المستخدم بتشغيلها لثانى مرة أو أكثر. ولكن هذا الاشتراط يمكن ان يؤدى الى مشاكل كبيرة. حاول تشغيل اثنين من ال Microsoft Word instances - عندما فعلت هذا، تلقيت رسالة الخطأ ان وثيقة ال Normal.dot يتم استخدامه بواسطة تطبيق آخر وغالباً، هذه هى الحالة الاصلية لل Web.

وبدلاً من ذلك، يجب ان تقرر هل من الملائم لتطبيقك ان يتم تشغيله على انه حالة فردية، أو على انه حالات متعددة عندما يعيد المستخدمون بدء تشغيل البرنامج. والحيلة فى عمل هذا ليس بسيطة ولكن ممكنة بواسطة تعديل ال code source لمشروع Delphi. اتبع هذه الخطوات لتجرب التقنية :

١- ابدأ تطبيقاً جديداً تريد ان تحفظ ملفاته على القرص بحيث تستطيع تشغيل ملف ال .exe من ال Windows Explorer. (لا يمكنك تشغيل program instances من داخل Delphi). قم بإنشاء دليل للمشروع. احفظ ملفات ال unit والمشروع باستخدام اسماء الملفات الافتراضية لها.

٢- اختر View|Project Source لفتح مشروع البرنامج كما ترى، ان المشروع بالفعل يعتبر برنامج ال Pascal الرئيسى للبرنامج.

٣- أضف Dialogs و Windows units لإمر ال uses لملف المشروع. لقد فعلت هذا تحت ال Forms unit مباشرة، ولكن الموضع لا يهم غالباً.

٤- قم بتعديل عبارات المشروع بين ال begin وال end باستخدام القائمة (٢١-١٣) كمرشد لك ان ال code المذكورة تفترض ان ال form الرئيسية للتطبيق تحمل اسم TForm1. اذا غيرت اسم ال form، يجب ان تقوم بتحديث كل المراجع المشيرة إليها.

٥- احفظ المشروع لتنشئ الدليل الخاص به، واضغط Ctrl+F9 لل compile. اغلق كل الملفات. انتقل الى ال Windows Explorer، وقم بتشغيل ملف ال Project1.exe. دون اغلاق النافذة، قم بتشغيل البرنامج مرة اخرى- فى هذه المرة سوف ترى رسالة ان البرنامج فى حالة تشغيل الآن. اغلق ال dialog الرسالة للعودة بصورة تلقائية الى حالة البرنامج الاولى.

**ملحوظة:** وجدت انه يجب ان اغلق ملفات التطبيق قبل ان اتمكن من تشغيل اثنين من حالات البرنامج. وقد يكون هذا نتيجة لاستدعاء ال FindWindow عندما كانت ملفات مصدر التطبيق مفتوحة فى Delphi لان class النافذة لل form الموجودة فى Delphi تحمل اسم TForm1.



القائمة (٢١-١٣): استخدم هذه القائمة كمرشد لك  
لمنع مشروع من التشغيل عدة مرات

```
program Project1;
```

```
uses
  Forms,
  Dialogs,
  Windows,
  Unit1 in 'Unit1.pas' {Form1};
```

```
{ $R *.RES }
```

```
begin
  HPrevInst := FindWindow('TForm1', nil);
  if HPrevInst = 0 then
  begin
    Application.Initialize;
```

```
Application.CreateForm(TForm1, Form1);
Application.Run;
end else
begin
ShowMessage('Application is already running');
Windows.SetFocus(HPrevInst);
Windows.SetForegroundWindow(HPrevInst);
end;
end.
```

فى الماضى، كان الـ Windows يحدد الـ HPrevInst للـ program instance handle الموجودة بالفعل. لم يعد هذا موجوداً، ولكن، يمكننا ان نجعل الخطوة مزدوجة باستدعاء الـ FindWindow كما تم التوضيح. ان الـ parameter الاول الذى يتم تمثيره الى الـ FindWindow هو اسم الـ class والـ parameter التالى، الذى يستخدم هنا، يمثل الـ Caption النافذة. هذا يفترض ان اسم الـ class يعرف التطبيق مطلقاً وان الـ form object لم يتم إنشاؤه بعد بواسطة التطبيق.

ويعدل المشروع المعدل الـ HPrevInst، والتى اذا كانت مساوية لصفر تشير الى ان هذه هى المرة الاولى التى يتم فيها تنفيذ البرنامج. ان تشغيل التطبيق يبحث عن النافذة، ويحدد الـ HPrevInst بـ instance handle التطبيق. اذا كان الـ handle غير صفري، فإن عبارات الـ else الخاصة بالمشروع تعرض رسالة وتستدعى الـ Windows API SetFocus function لإعطاء الحالة الاولى input focus. ان عرض نافذة الحالة يتطلب ايضاً استدعاء الـ SetForegroundWindow كما هو موضح.

لاحظ ان الـ SetFocus والـ SetForegroundWindow يتم استدعاؤهما بالإشارة مباشرة الى الـ Windows unit، حيث يتم تعريف كلتا الـ functions. وهذا ليس ضرورياً، ولكن لان Delphi components أخرى لها الـ SetFocus methods، فأن تسبق الاستدعاءات بالـ Windows يضمن ان البرنامج يستدعى الـ methods الصحيح. يمكنك ان تفعل هذا مع أى unit لها الـ methods مزدوجة وأسماء الـ object متداخله.



**فكرة:** للانتقال بهدوء الى program instance الاولى ، احذف عبارة  
ال ShowMessage .



### بعض ملفات مجموعة ال DOS النافعة:

هل مازالت تكتب اوامر على ال DOS prompt كما افعل انا عادة ؟ إننى  
اكتب دائماً ملفات مجموعة batch files لأداء عمليات متنوعة فى نافذة DOS .  
وفيما يلى بعد الملفات التى وجدتها نافعة للغاية وبخاصة مع Delphi .

**على القرص المدمج:** توضح القائمة (٢١-١٤) ملف ال Clean.bat  
من على القرص المدمج فى دليل ال Source . فيما عدا الملفات الهامة  
فى مشروع Delphi . تعتبر القائمة (٢١-١٥) جزءاً من ملف ال  
Cleanall.bat الذى يمكنك تشغيله لحذف الملفات التى لاداعى لها بعد ان تقوم بـ  
compile البرامج المثالية لهذا الكتاب . لاستخدام هذه البرامج ، انسخها على  
الدليل الذى يحتوى على الادلة الفرعية لمشروعك . قم بتشغيل ال Cleanall لمسح  
كل الادلة . أو ، قم بتشغيل ال Clean لمسح ملفات فردية مثل هذا .



#### القائمة (٢١-١٤) : Clean.bat

```
@echo off
rem
rem CLEAN.BAT
rem Delete files created by Delphi
rem Called by CLEANALL.BAT
rem Input %1==subdirectory where clean.bat is stored
rem
rem ex. call clean Polyflow
rem
if "%1"==" " goto ERROR
if exist %1\NUL goto CONTINUE
goto ERROR
:CONTINUE
cd %1
echo Cleaning %1
```

## الباب الحادي والعشرين : تطوير مهارات Delphi الخاصة بك

```
rem
rem Add new files to delete here:
rem
if exist *.bak del *.bak
if exist *.* del *.*
if exist *.dsk del *.dsk
if exist *.dsm del *.dsm
REM if exist *.exe del *.exe
rem
rem Go back to directory that has clean.bat
rem
:REPEAT
cd ..
if exist clean.bat goto END
goto REPEAT
:ERROR
echo.
echo Error cleaning: %1
echo No such directory
pause
:END

القائمة (٢١-١٥) Clean.bat (القائمة جزئية-الباقى على القرص)
@echo off
rem
rem cleanall.bat-call clean.bat for all source directories
rem
echo working...
if exist *.bak del *.bak
call clean aboutex
call clean addpage
call clean barchart
call clean bitview
call clean bitview2
```

```
...
echo done
:END
```

لضمان السلامة ، لقد اشرت بتعلق الى التعليمات الموجودة فى ال Clean.bat والتي تسمح ملفات ال .exe . اذا قمت بتشغيل هذا السطر ، كن شديد الحذر من ان تسمح كل التطبيقات الموجودة على محرك قرصك الصلب . لا يجب ان تقوم بتشبيت ال Clean أو ال Cleanall فى دليل ال Path العمومى . كذلك قد تريد ان تحتفظ بنوع أو آخر من الملفات . انظر الباب الاول ، فصل "امتدادات اسم الملف" لتعرف وصف الملفات التى يقوم Delphi بإنشاءها .

### ال parameters الافتراضية:

يضيف Delphi 4 افتراضية الى ال parameters Object Pascal . وهذه الميزة ، والمقتبسة من ال C++ ، توفر قيم افتراضية لل function وال parameter procedure على سبيل المثال ، افترض ان لديك procedure له ثلاثة parameters معرفة كمايلى :

```
procedure CallMe(A: Integer; B: Double; C: String);
```

لاستدعاء هذا ال procedure فى عبارة program ، يجب ان توفر قيم argument ، parameters الثلاثة :

```
CallMe(10, 3.14159, 'Message from a flounder');
```

ولكن ، افترض ان بعض الاستدعاءات لل procedures لم تكن تحتاج الى توفير قيم لل parameter ، فبدون ال parameters الافتراضية ، يجب عليك كتابة code مثل هذه :

```
CallMe(10, 2.2, "");
CallMe(20, 3.3, "");
CallMe(30, 4.4, "");
```

فى كل حالة ، إنك تحتاج أن تقدم null string لل CallMe ، بالرغم من أن قيم ال parameter الذى يتم استخدامه . إن قيم ال parameter الافتراضية تساعد على تبسيط هذا النوع من ال code . قم بتعريف parameter افتراضى كما يلى :

```
procedure CallMe(A: Integer; B: Double; C: String = '');
```

إن القيمة الافتراضية المعنية للـ C parameter تخبر الـ compiler أن يستخدم هذه القيمة إذا لم يتم توفير قيمة واضحة. مع هذا التعريف في موضعه، يمكن أن يتم الآن كتابة الاستدعاءات الثلاثة السابقة للـ procedure كما يلي:

```
CallMe(10, 2.2);
```

```
CallMe(20, 3.3);
```

```
CallMe(30, 4.4);
```

إن القيم الافتراضية يمكن تعيينها لأي parameter، ولكن يجب أن تأتي في آخر التعريف. لا يمكنك مثلاً استخدام تعريف مثل هذا:

```
procedure X(A: Integer = 0; B: Double); // ???
```

هذا لا يؤدي إلى الـ compile لأن قيم الـ parameter الافتراضية يجب أن تأتي في النهاية دائماً. في هذه الحالة، يمكنك أن تعكس ترتيب الـ parameters أو توفر قيمة افتراضية للـ B أيضاً:

```
procedure X(A: Integer = 0; B: Double = 3.14159);
```

إن الـ parameters الافتراضية تبسط بعض أنواع العبارات، وتساعد على ضمان أن يكون لكل الـ parameters قيم معروفة. والعيب الوحيد من استخدام قيم الـ parameters هي الطريقة التي تشوش بها على الـ arguments الفعلية التي يتم تمريرها إلى الـ functions و procedures. على سبيل المثال، إن الاستدعاءات الثلاثة السابقة للـ CallMe لا تعطى إشارة عن أن الـ null string argument قد تم تمريرها كـ argument ثالثة سرية.

### :Method overloading

يضيف Delphi 4 ميزة أخرى معروفة ومستعارة من الـ C++ وهي الـ method overloading أو التحميل الزائد للـ method. هذا يسمح لإثنين أو أكثر من الـ procedures أو الـ functions أن يكون لهم أسماء متطابقة، ولكن يعملوا كنظم فرعية مختلفة والقاعدة البسيطة هي: أن الـ overloaded method يجب أن

يختلف فى نوع parameter واحد على الأقل . ومادمت تلتزم بهذه القاعدة ،  
يمكنك أن تزيد تحميل نفس ال method كلما دعت الحاجة .

إن زيادة تحميل ال method يساعدك على إنشاء code مفهومة تتوافق مع  
الاهداف المختصة بال object ، مثلاً ، تطوير مجموعة من ال procedures كلها  
تحمل اسم ShowItem والتي تعرض نوع من الشكل البيانى . يمكنك عندئذ كتابة  
عبارة مثل :

ShowItem(X);

حيث أن ال X يمكن أن يكون لها شكل معروف لأنواع عديدة مختلفة . وهذا  
يعتبر أسهل من كتابة procedures متعددة تحمل اسماء ShowIntegerItem ،  
ShowGraphicsItem ، ShowImageItem ، الى آخره .

### مثال على التحميل الزائد لل Method:

وكمثال آخر أكثر اكتمالاً ، افترض أنك تريد تطوير procedure لبدء  
arrays ديناميكية من أنواع مختلفة . أولاً ، إنك تعرف أنواع ال array الديناميكية  
كما يلى :

type

TIntArray = array of Integer;

TFloatArray = array of Double;

TStringArray = array of String;

بعد ذلك ، قم بتعريف متغيرات لل arrays الثلاثة :

var

IntArray1: TIntArray;

FloatArray1: TFloatArray;

StringArray1: TStringArray;

وجزئيتى ال code هذه توضحان الاستخدام العملى لتعريف أنواع ال  
array . إننا نحتاج الى إنشاء arrays فعلية للأنواع الثلاثة ، ونحتاج الى تعريف  
procedure parameters لنفس هذه الأنواع . ويتم تعريف ال parameters  
الثلاثة هكذا (داخل قطاع ال class public مثلاً) :

```
public
  procedure InitIntArray(A: TIntArray; V: Integer);
  procedure InitFloatArray(A: TFloatArray; V: Double);
  procedure InitStringArray(A: TStringArray; V: String);
```

إن كل procedure يتم تنفيذه لتعيين قيمة V لكل موضع array، وبهذا يحدد كل عناصر ال array بقيمة معروفة. يمكن تنفيذ ال procedure الأول كما يلى :

```
var
  I: Integer;
begin
  for I := 0 to High(A) do
    A[I] := V;
end;
```

بعد كتابة هذا الجزء من ال code، يمكنك استدعاء procedures لإنشاء ال array مثل هذه :

```
InitIntArray(IntArray1, 0);
InitFloatArray(FloatArray1, 1.0);
InitStringArray(StringArray1, 'Msg');
```

هذا يحقق الهدف ولكنه فوضوى بعض الشيء مع ال method overloading، يمكن تحديد اسم procedure واحد لكل أنواع عند انشائها ال array. لكى تفعل هذا، قم بتغيير كل اسماء ال procedure حتى تصبح واحدة، وأضف الكلمة الأساسية overload بعد التعريف :

```
procedure InitArray(A: TIntArray; V: Integer); overload;
procedure InitArray(A: TFloatArray; V: Double); overload;
procedure InitArray(A: TStringArray; V: String); overload;
```

ان procedure الثلاثة من الممكن ان يكون لها نفس الاسماء لان تعريفها تختلف فى نوع procedure واحد على الاقل. قم بتنفيذ كل procedure (يجب ان تقوم بتنفيذ ال procedures الثلاثة منفصلة) كما سبق، باستخدام نفس

التعريفات الموضحة هنا . لاستدعاء ال procedures ، استخدم code مثل هذه :

InitArray(IntArray1, 0);

InitArray(FloatArray1, 1.0);

InitArray(StringArray1, 'Msg:');

لا يوجد سحر فى هذا الأمر . فداخلياً، مازال هناك ثلاثة procedures منفصلة . ولكن، بدلاً من التمييز بينها بالاسم، مع ال method overloading ، من الممكن التفرقة بين ال procedures بواسطة البيانات التى تم تمريرها كـ arguments هذا يوضح ال code، ويجعل من السهل إضافة InitArray procedures جديدة لأنواع بيانات اخرى .

**ملحوظة:** ان ال overloaded method يجب ان يكون له نوع parameter واحد على الأقل يختلف عن ال methods الاخرى التى لها نفس الاسم أو يجب ان يكون ال methods مختلفة من ال parameters . ان نوع وعدد ال parameters هو الذى يهم .

Note

### التحميل الزائد للـ method والـ parameters الافتراضية:

يمكنك استخدام قيم parameter افتراضية مع ال overloaded methods . على سبيل المثال، يمكن تعيين قيمة افتراضية للـ V parameter فى التعريفات السابقة باستخدام تعريفات مثل :

procedure InitArray(A: TIntArray; V: Integer = 0); overload;

procedure InitArray(A: TFloatArray; V: Double = 1.1);  
overload;

procedure InitArray(A: TStringArray; V: String = "");  
overload;

والآن، من الممكن بدء ال arrays للأنواع المعرفة بعبارات مثل هذه :

InitArray(IntArray1);

InitArray(FloatArray1);

InitArray(StringArray1);

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

ان التحميل الزائد للـ methods، مع قيم الـ parameters الافتراضية، ينتج عادة code مثل هذه، بسيطة وواضحة. ومن الاسهل ان تتذكر اسماء الـ method دون الاضطرار الى مراجعة تعريفها.

على الجانب الآخر، ان دمج الـ Method overloading مع قيم الـ parameter الافتراضية يمكن ان يؤدي الى code محيرة:

```
procedure F(Height: Integer; D: Double = 0.0); overload;  
procedure F(Width: Integer); overload; // ???
```

هذا يضع الـ compiler فى حيرة لأنه ليس من الممكن دائماً تحديد أى procedure تنوى العبارة استدعاؤه هذه العبارة، مثلاً:

```
F(10); // ???
```

لا يتم الـ compile. وبسبب قيمة الـ procedure الافتراضية فى التعريف الأول، يمكن ان تستدعى العبارة أى من الـ procedures. للقضاء على المشكلة، يجب عليك الا تعين قيمة procedure افتراضية للـ D، أو يجب ان تعيد تسمية الـ procedure الثانى (بـ F2، مثلاً).

### : Method overloading and class constructors

ان الـ Method overloading له قيمة عالية فى إنشاء class constructors. فى النسخ السابقة من الـ Object Pascal، كان يجب تسمية الـ constructors باسماء مختلفة. على سبيل المثال، يمكن لـ class نموذجية فى الـ VCL ان تعرف الـ constructors التالين:

```
constructor Create(AOwner: TComponent); override;  
constructor CreateNew(AOwner: TComponent; Flag:  
Integer);  
virtual;
```

مع الـ method overloading، يمكن ان للـ constructor الثانى ان يسمى Create لان له parameter واحد زائد عن الـ constructor الأول. لزيادة تحميل كلا الـ constructors، قم بتغيير إسمهما لـ Create وأضف الكلمة الاساسية overload. والتعريفات الجديدة تصبح:



```

constructor Create(AOwner: TComponent); overload;
override;
constructor Create(AOwner: TComponent; Flag: Integer);
virtual; overload;

```

هذا يؤدي الى تبسيط إنشاء الـ class objects لان الـ constructors الصحيح يتم الآن بناء على استخدامه بدلاً من الاسم الظاهري له . على سبيل المثال . هذه العبارة تنشئ objects من الـ class :

```

Obj1 := TForm.Create(AOwner);
Obj2 := TForm.Create(AOwner, 1);

```

ان العبارة الأولى تستدعي الـ constructor الأول ، والذي يسيطر على Create method (overrides) الموروث الخاص به . والعبارة الثانية تستدعي الـ constructor الثانى لأنها تقدم argument عدد صحيح للـ Flag parameter الخاص بهذا الـ constructor .

### : Method overriding versus overloading

ان احدى الصعوبات التى قد تواجهها مع الـ overloaded methods هي عندما يخفى تعريف الـ methods الاصلى والموروث له نفس الاسم . ولنقل هذا بعبارة اخرى ، هناك صراع قوى بين الـ overriding على الـ methods موروث والـ overriding للـ methods . ان online help الخاصة بـ Delphi تعطى امثلة مشابهة لمايلي . تقوم code بتعريف classes . ها هي الأولى :

```

FirstClass = class
public
  procedure X(I: Integer); virtual;
end;

```

والـ class الثانى مشتقة من الأولى :

```

SecondClass = class(FirstClass)
public
  procedure X(S: string);
end;

```

وهذه الانواع من التعريفات يتم استخدامها عامة لاستبدال (X) procedure في المثال) في class مشتقة ان البرنامج المستبدل في ال SecondClass يعرف parameter من نوع مختلف عن ال methods الموروثة الخاص به. ولذلك، فإن الاستدعاء الأول فقط من الاستدعائين التاليين للـ X يتم ال compiles :

```
var
  AnObject: SecondClass;
begin
  AnObject.X('Testing');
  AnObject.X(123); // ???
end;
```

ويفشل الاستدعاء الثاني للـ X في ال compile لان إعادة التعريف للـ method الموروثة يخفي التعريف الأصلي الذي تعرف على أنه Integer parameter. والـ overloading Method تجعل من الممكن حل هذا النوع من الصراعات. يتم تعريف الـ classes الجديدة كمايلي :

```
FirstClass = class
public
  procedure X(I: Integer); virtual; overload;
end;
SecondClass = class(FirstClass)
public
  procedure X(S: string); overload;
end;
```

ان الاختلاف الوحيد هو استخدام الكلمة الاساسية overload. ولان الـ procedures يعتبران overload فان استدعاء الـ procedure للـ X الآن يتم الـ compile لان الـ compiler يمكن أن يميز بين الـ procedure من خلال انواع الـ parameter المختلفة لهما.

ولان وراثه method وتغيير الـ parameters الخاصة به، وليس اسمه، يؤدي الى إخفاء التعريف الاصلى - في هذا المثال، في الـ SecondClass يخفي

ال X فى ال FirstClass- فإن ال compiler يصدر تحذيراً. لإزالة هذا التحذير، استخدم الكلمة الأساسية reintroduce فى التعريف التالى هكذا:

```
SecondClass = class(FirstClass)
public
  procedure X(S: string); reintroduce;
end;
```

هذا يخبر ال compiler انك تنوى التعتيم على ال X الاصلية وانك لا تتوقع ان يقوم التعريف الجديد لـ method ال overload.

### : Method overloading and forward declarations

ان كلمة التعريف الأساسية forward والخاصة بال Object Pascal يتم استخدامها الآن بطريقة مختلفة قليلاً بما ان اللغة method overloading. فى الماضى، كان يمكنك ان تعريف procedure أو function forward، مما يشير الى ان تنفيذها يأتى بعد ذلك فى ملف ال source code على سبيل المثال،

```
// Declare forward procedure
procedure X(D: Double); forward;
... // Code may call X here
// Implement the procedure
procedure X;
begin
  // Code for X
end;
```

ان تعريف ال forward يعطى ال compiler كل المعلومات التى يحتاجها ليتعامل مع العبارات لإستدعاء ال X. ولكن، اذا كنت تنوى ان overload ال X بـ procedure ثانى متميز يسمى X ايضاً دون parameters، فإن ال code السابقة تكون محيرة. ان إضافة الكلمة الأساسية overload للتعريف الاول يجعل ال compiler ينشئ procedure جديد منفصل يحمل اسم X وبلا parameters، وهو ليس ما تريد بالطبع. لحل هذا الصراع، يسمح ال Object Pascal الآن لك

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

إعادة تعريف ال forward procedure مع parameters . على سبيل المثال، ل overload ال X مع نسخة بلا parameters ، ولتنفيذ نسخة X مع ال Double parameter ، يمكنك كتابة تعريفات مثل هذه :

```
// Declare overloaded forward procedure
procedure X(D: Double); overload; forward;

...
// Implement an overloaded, distinct procedure
procedure X; overload;
begin
end;
// Implement forward declaration
procedure X(D: Double); overload;
begin
end;
```

هذا لن يكون ممكناً إذا لم تكن تنفيذات ال forward لا يمكن ان يكون لها تعريف لل parameter . ان ال X الذى ليس له parameters فى هذا المثال هو procedure منفصل عن ال forward procedure X الذى يعرف ال Double parameter.

### arrays ديناميكية:

يضيف Delphi 4 arrays ديناميكية لل Object Pascal . ان ال array الديناميكي يعرف نوعه ، وليس حجمه . ال array الفعلى يتم إنشاؤه فى وقت التشغيل بعبارة برنامج . إنشئ array ديناميكياً مع ال static array العام لل Pascal . على سبيل المثال ، ان تعريف المتغير هذا :

```
var
```

```
IntArray : array[1 .. 100] of Integer;
```

ينشئ ال array فى ذاكرة من ١٠٠ عدد صحيح . ك array ديناميكى ، يتم تعريف ال IntArray أولاً دون نطاق من قيم الفهرس فى الاقواس :

var

IntArray : array of Integer;

. هذا يشير الى ان نوع ال IntArray هو array من قيم ال Integer .  
لاستخدام ال array يجب ان تقوم عبارة program بتخصيص مساحة لل array  
لحمل هذه القيم .

والطريقة الثانية ، والمستحبة ، لإنشاء arrays ديناميكية هي باستدعاء ال  
SetLength procedure ، والذي كان يتم استخدامه فى الماض لتغيير اطول  
ال strings فقط . استخدم ال SetLength لبدء array ديناميكى كمايلى :

begin

SetLength(IntArray, 100);

...

end;

بعد تخصيص مساحة array ، يمكنك استخدامه كما تفعل تماماً مع النوع  
الساكن . كما فى ال strings ، انك لا تحتاج الى تحرير المساحة المخصصة . ولكن ،  
يمكنك إزالة مساحة ذاكرة ال array الديناميكى بتحديد طوله بصفر بالعبارة :

begin

SetLength(IntArray, 0);

...

end;

والاختيار الآخر هو ان تعين nil لتغير ال array يمكن تبسيط ال code السابقة  
الى مايلي :

IntArray := nil;

يمكنك تعريف array ديناميكى من أى نوع . على سبيل المثال ، هذه العبارة  
تعرف array ديناميكى من نوع ال strings :

var

StrArray : array of string;

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

يمكنك أيضاً تعريف arrays ديناميكية متعددة الابعاد، وهى حرفياً "arrays of arrays." على سبيل المثال، التحديد الساكن :

```
var
```

```
Grid: array[1 .. 10, 1 .. 25] of Double;
```

ينشئ static array ثنائى الابعاد ذا عشرة صفوف و ٢٥ عمود من قيم ال Double floating-point. ك array ديناميكى، يمكن تعريف ال array كمايلي :

```
var
```

```
Grid: array of array of Double;
```

```
begin
```

```
Grid := New(Grid, 10, 25);
```

```
...
```

```
end;
```

أو بدلاً من استخدام ال New، يمكنك استخدام SetLength كمايلي :

```
SetLength(Grid, 10, 25);
```

وبأى الطريقتين، يتم تخصيص مساحة ال Grid ثنائى الابعاد، ٢٥×١٠، لقيم Double floating-point :

وعلى عكس ال arrays static، لا يجب ان يكون ال arrays الديناميكية مستطيلة. بعبارة اخرى، ان صفوف واعمدة ال arrays الثنائى الابعاد والأكثر من ذلك لا يجب ان تكون من نفس الحجم. بعد تحديد الطول أو استخدام ال New كما تم التوضيح، يمكنك اعادة تخصيص أى صف أو عمود مع code مثل :

```
begin
```

```
SetLength(Grid, 10, 25);
```

```
SetLength(Grid[2], 12);
```

```
...
```

```
end;
```

ان العبارة الاولى تخصص لل Grid array ٢٥×١٠ مستطيل من قيم ال Double. والعبارة الثانية تعيد تخصيص الصف الثالث لهذا ال array، مما يقللها

الى ١٢ قيمة . من الممكن ايضاً تخصيص الصفوف للـ arrays متعددة الابعاد فقط .  
على سبيل المثال ، العبارة التالية تنشئ array ، حيث تختلف الاعمدة فى الحجم من  
عنصر واحد الى أى عدد (n) من العناصر . اختبر هذا الـ code :

```
var
  Triangle: array of Integer;
  I: Integer;
begin
  SetLength(Triangle, 10); // Allocate 10 rows
  for I := 1 to 10 do
    SetLength(Triangle[I], I);
  ...
end;
```

بغض النظر عن كيفية تخصيص مساحة للـ arrays الديناميكية ، فهى دائماً  
تكون مفهومة من صفر الى arrays اصغر حجماً فى عدد العناصر . لضمان  
السلامة . استخدم الـ Low و High للحصول على الروابط القانونية  
للـ array الديناميكي . على سبيل المثال ، فى هذه الجزئية :

```
var
  IntArray: array of Integer;
  I: Integer;
begin
  SetLength(IntArray, 100);
  for I := Low(IntArray) to High(IntArray) do
    IntArray[I] := I;
  end;
```

للـ array الديناميكي ، يقوم الـ Low(X) بإدخال صفر ، لذا فالـ Loop  
السابقة يمكن كتابتها بسلامة كهذا :

```
for I := 0 to High(IntArray) do
  ...
```

## الباب الحادى والعشرين : تطوير مهارات Delphi الخاصة بك

وهذه تعتبر افضل طريقة دائماً لمنع خطأ المدى - وهو فهرسة عنصر array غير موجود بعد أعلى منطقة مخصصة. وتقوم ال High function بارجاع عدد العناصر فى ال array ناقص واحد.

**تحذير:** لا تقم بطرح واحد من ال High(X). تقوم ال function بارجاع عدداً يكون اقل من عدد العناصر فى ال arrays الديناميكي X. كن حذراً عند نسخ arrays ديناميكية - فهي لا تعمل كال strings. على سبيل المثال، اذا كان لديك arrays من نفس النوع والحجم تماماً، يمكنك تعيين array واحد للآخر، كما هو موضح هنا:



```
var
  A, B: array of Double;
begin
  SetLength(A, 100);
  ... // Use array A
  B := A; // Copy A to B which must be equal types
  ... // Use arrays A & B
end;
```

ال code يعرف اثنين من ال arrays الديناميكية لقيم Double floating-point. ال arrays الأول، A، يخصص مساحة فى الذاكرة، ويتم استخدامه. ان عبارة التعيين تنسخ ال arrays A الى B، ولكن، لان A و B هي مجرد (pointers)، فإن العبارة تجعل A و B يخاطبا نفس الذاكرة. فهي لا تشيئ arrays منفصلين. ولكن، بخلاف ال strings، فإن ال arrays الديناميكية لا تستخدم ال copy-on-write algorithms.

ان ال Reference counting تحافظ على عدد ال pointers address التى لها نفس الذاكرة. فقط عندما يتم التخلص من ال pointer النهائي يتم إدخال الذاكرة المخاطبة للمجموعة. ويؤكد ال copy-on-write algorithms ان، اذا حدث تغيير لذاكرة تم مخاطبتها بواسطة اثنين أو أكثر ال pointers، نسخة جديدة من الذاكرة يتم عملها قبل السماح بهذا التغيير. وبذلك، اذا نسخت strings واحد



الى الآخر، فإن strings واحداً فقط يوجد فى الذاكرة حتى تقوم بالتغيير الى ال strings المخاطبة. وهذه ليست الحالة مع ال arrays الديناميكية. على سبيل المثال، فبعد عبارة التعيين فى المثال السابق، فهذا ال code:

```
A[1] := 3.14159;
```

تقوم ايضاً بتغيير B[1] الى قيمة ال pi.

يمكن تغيير احجام ال array الديناميكي فى وقت التشغيل. فى الواقع، هذه واحدة من المزايا الاساسية لل arrays الديناميكية. فهذه ال code، مثلاً، تنشئ array من قيم ال Double تستخدم ال array، ثم تستدعى ال Copy function لإعادة تحديد حجم ال array:

```
var
  Values: array of Double;
begin
  SetLength(Values, 100);
  ... // Use array Values here
  Values := Copy(Values, 500);
  ... // Use larger array here
end;
```

فى هذا ال code، ال Values تعتبر array ديناميكي لقيم Double floating-point. ان العبارة الأولى فى البرنامج تعد ال values لحمل مائة عنصر. والعبارة الثانية تستدعى ال Copy function لإنشاء array اكبر يتكون من خمسمائة عنصر، وتنسخ القيم الموجودة لل array الجديد. بعد ذلك، يتم ترك ال array الاصلى.

**ملحوظة:** اثناء الاستدعاء لل Copy، يوجد ال arrays فى الذاكرة-واحداً من الحجم الاصلى وواحداً من الحجم الجديد. اذا احتجت الى توفير استخدام الذاكرة، فكن متبهاً لهذا عند استدعاء ال Copy لإعادة تخصيص مساحة الذاكرة.



وفى حالة الحاجة ل arrays فعلية كثيرة من نفس النوع، يمكنك إنشاء نوع بيانات array ديناميكي باستخدام code مثل التالية:

```
type
  TIntArray = array of Integer;
  TFloatArray = array of Single;
var
  IntArray1: TIntArray;
  IntArray2: TIntArray;
  FloatArray1: TFloatArray;
begin
  SetLength(IntArray1, 100);
  SetLength(IntArray2, 200);
  SetLength(FloatArray1, 300);
  ...
end;
```

يتم تعريف نوعين من الـ array الديناميكي : TIntArray و TDoubleArray .  
يتم تحديد المتغيرات عندئذ باستخدام انواع البيانات والمساحة المخصصة باستخدام الـ  
SetLength . وميزة هذه التقنية هي سهولة التعديل لنوع الـ array . على سبيل المثال ، فى الـ  
code السابقة ، يمكنك تغيير نوع الـ TFloatArray هكذا :

```
type
  ...
  TFloatArray = array of Double;
```

هذا هو التغيير الوحيد المطلوب لتعديل كل الـ arrays الديناميكية الفعلية من  
نوع الـ TFloatArray من الـ single-precision الى الـ double-precision  
لعناصر الـ floating-point .

### ملحوظة اخيرة:


هذا هو آخر فصل فى الكتاب ، ولكن ليس آخر الموضوع . فى الواقع ، ان خبرتك  
مع Delphi فى بدايتها . ان شعبية Delphi تزداد يوماً بعد يوم . فإن مطوري البرمجيات  
على مستوى العالم يتبعون Delphi ويكتشفون المزيد عن قيمة الـ objects-oriented  
objects اذا قرأت هذا الكتاب ، فإن لديك الآن قاعدة ثابتة فى تقنيات تطوير Delphi .  
أتمنى ان تكون قد استمتعت بالرحلة . والآن جاء دورك قم بالبحث فى online help ،  
وقم بتعليم نفسك موضوعات جديدة بكتابة برامج اختبارية قصيرة .

## How to Use the CD-ROM

The CD-ROM inside the back cover contains all sample projects described in this book. Copy the files from any CD-ROM directory to an empty directory on your hard drive and then open the copied project with Delphi's File|Open... or File|Open Project... commands. Browse to any directory and open the .dpr file—there is only one in most project directory paths. If you don't see a .dpr file, make sure the Open dialog's Files of Type box displays Delphi file (\*.pas, \*.dpr). Not all sample directories, however, contain projects. For these, there are no .dpr files to open.

### Using the Listings

All listings in the book are located in named directories on the CD-ROM. None of these files is compressed. To use the files, copy any subdirectory from the Source directory to your hard drive. Or, if you have the room, copy the entire Source directory to a hard drive letter such as C:\ or D:\.

 **Note** Some readers have requested that I organize sample applications by chapter number. This is a useful convention in some cases, but in Delphi 4 Bible, several different chapters refer to the same applications. For this reason, all sample applications are stored in directories named the same as their project names. For example, the SysColor demonstration's files are located in the Source\SysColor directory.

Although most directories contain full Delphi projects, some such as Misc contain .pas or other files. Use Delphi's File|Open... command to view these files.

### Running and Compiling Programs

All programs are compiled. If you just want to run them, use the Windows Explorer program to execute any .exe code file. You may execute most programs directly from the CD-ROM. However, those programs that write to disk files must first be copied to your hard drive or another writeable device.

To compile a program, you must first copy its directory to your hard drive. If you attempt to compile a program directly to the CD-ROM, Delphi will display an error message when it attempts to write to the .exe code file. If, after copying a directory to your hard drive you still have trouble compiling, the target files might be marked Read-Only. See "Unmarking Read-Only Files" next for more help.

### Unmarking Read-Only Files

Copying CD-ROM files might set their Read-Only attribute flags, which will prevent writing to the .exe and .dcu code files. If, when compiling a program you receive an error that a file is in use or marked read-only, follow these steps to correct the problem:

1. Copy the Source directory from the CD-ROM to your hard drive directory C:\, D:\, or other.
2. Open a command-line prompt (MS-DOS) window.
3. Enter C: or D: (or another letter) to change to the Source drive.
4. Enter the command CD \Source to change to the Source directory.
5. Type the command ATTRIB -r \*.\* /S to turn off all Read-Only flags.
6. If you receive a "sharing violation" error in Step 4, exit Delphi and try Step 4 again.

## الفهرس

٧	محتويات الكتاب
٩	مقدمة
١٥	الجزء الأول : مقدمة
١٧	الباب الأول : معلومات حول Delphi4
١٧	لمحة تاريخية
٢٩	تطبيقات Delphi المطورة
٤٠	برمجة الـ Component
٤٦	(Code Insight)
٥٥	Module Explorer
٥٩	الملفات وامتدادات اسم الملف
٦٣	ملفات الـ Unit
٧١	Project Manager
٧٩	ملخص
٨١	الباب الثاني : معلومات حول الـ Visual Components
٨١	الـ Visual Component Library
٨٢	Visual component categories
٨٥	الأمثلة التطبيقية
٨٥	Standard Components
١٠٤	Additional Components و Dialogs Components
١٠٦	التطبيق BitView
١١٣	System Components
١١٨	Win32 Components
١٢٦	الملخص
١٢٩	الباب الثالث : معلومات أولية حول الـ Forms
١٣٠	Forms as Components
١٣٩	Forms Templates
١٤٩	استخدام الـ Dialog للنافذة الرئيسية

١٥١	إغلاق النافذة
١٥٣	Form Frameworks
١٧٢	مقدمة حول الـ Data Modules
١٧٨	الـ Splitter Component
١٨٣	ملخص
١٨٧	الجزء الثانى: واجهة تطبيق المستخدم
١٨٩	الباب الرابع: برمجة لوحة المفاتيح والفأرة
١٨٩	حول الجزئين الثانى والثالث
١٩٢	على لوحة المفاتيح
٢١٤	Mouse Traps
٢٢١	Custom mouse cursors
٢٢٥	الرسم بالفأرة
٢٣١	ملخص
٢٣٣	الباب الخامس: بناء القوائم
٢٣٤	Components
٢٣٤	Pop-Up Menus
٢٤٧	Floating Pop-up Menus
٢٤٩	Dynamic Menus
٢٦٩	Portable Menus
٢٧٣	Menu Designer افكار عن الـ
٢٧٤	ActionLists
٢٨١	ملخص
٢٨٥	الباب السادس: Attaching Buttons and Check Boxes
٢٨٦	Components
٢٨٨	أزرار اساسية
٢٩٢	ازرار ملونة
٣٠٤	مجموعات الأزرار
٣١٥	Spin Buttons
٣٢١	الـ Static Text أو (TStatic Text)

٣٢٦	Up-Down Button Controls
٣٣٤	ملخص
٣٣٧	الباب السابع: إنشاء الـ Coolbars، Toolbars، والـ Static Panels
٣٣٨	Components
٣٤٠	الـ Toolbar
٣٤٤	On/off (sticky) SpeedButtons
٣٦١	Status Panels
٣٧٧	الـ Coolbars
٣٩٦	الـ Status Bars
٤٠١	التطبيق Status
٤١٧	ملخص
٤١٩	الباب الثامن: تكوين الـ Lists
٤١٩	Components
٤٢٠	List Components
٤٢٤	Scrolling in listBoxes
٤٣٨	String and Other Lists
٤٦٨	ملخص
٤٧١	الباب التاسع: العمل مع الـ Single-Line Strings
٤٧١	Components
٤٧٢	Character Strings
٤٧٦	Built-In Text Dialogs
٤٨٨	Single-Line Text Components
٥٠٨	ملخص
٥١١	الباب العاشر: العمل مع نص متعدد السطور
٥١٢	Components
٥١٢	بفضل الـ Memos
٥٣٢	الـ Clipboard و Text
٥٣٣	التحريك بالـ ScrollBar
٥٤٧	الـ StringGrids
٥٥٧	ملخص

## الباب الحادى عشر: Navigating Directories and Files ... ٥٦١

٥٦٢	Components
٥٦٢	إنشاء Directory Dialog
٥٦٩	وضع dialogs الأدلة معاً
٥٧٢	تطوير ال Directory - Based Utilities
٥٧٥	ملفات Drag-and-Drop
٥٨٤	قراءة وكتابة ملفات ال .ini
٥٨٦	SysColorUtility
٦٠٩	ملخص

## الباب الثانى عشر: التعامل من خلال ال Dialog Boxes ... ٦١١

٦١٢	Components
٦١٤	Dialog Modes
٦١٦	Common Dialogs
٦١٧	Font and Color dialog
٦٢٦	Paged Dialogs
٦٢٨	ال Notebook
٦٣٥	إدخال صفحات اثناء وقت التشغيل
٦٣٧	من خلال ال Dialogs
٦٣٩	برمجة أوامر Find وال Replace
٦٤٤	Page Controls
٦٦٠	ال TabControl component
٦٦١	تحديد حجم النافذة
٦٦٥	إنشاء ال Docking Controls
٦٧٠	ملخص
٦٧٣	الجزء الثالث: التطبيق

## الباب الثالث عشر: تطوير تطبيقات الجرافيكية ... ٦٧٥

٦٧٦	Components
٦٧٧	معلومات حول ال Canvas
٦٨٣	الرسم والتصوير
٦٩٤	تقنيات برمجة الجرافيك

٧١٠	Picture Dialogs
٧١٠	الـ OpenPicture Dialog
٧١٩	الـ SavePictureDialog
٧٢١	Animations
٧٢١	الـ Animate component
٧٢٧	الـ MediaPlayer component
٧٤٠	ملخص
٧٤٣	<b>الباب الرابع عشر: تطوير تطبيقات الطباعة</b>
٧٤٣	Components
٧٤٤	Plain text
٧٤٩	إحصائيات الطباعة
٧٥٩	الـ TPrinter class
٧٥٩	خصائص الـ TPrinter
٧٦١	طباعة الجرافيك
٧٦٦	طباعة الـ bitmab، والأيقونات، والـ metafile
٧٦٧	طباعة أشكال الجرافيك
٧٩٣	ملخص
٧٩٥	<b>الباب الخامس عشر: تطوير تطبيقات الـ MDI</b>
٧٩٥	Components
٧٩٦	أسس برمجة الـ MDI
٨٠٤	الـ Child window
٨٠٤	الـ Child window ذات النوع الواحد
٨١٧	مجموعة من أوامر الـ MDI
٨١٨	العمل مع الـ Child window مختلفة النوع
٨٢٧	تقنيات MDI أخرى
٨٣٣	ملخص
٨٣٥	<b>الباب السادس عشر: التطوير مع الـ Clipboard، الـ DDE، والـ OLE</b>
٨٣٦	انتقالات البيانات عبر الـ Clipboard
٨٣٧	الـ TClipboard Class
٨٤٩	clipboard و Component object



٨٥٢	Dynamic Data Exchange
٨٦٥	اختبار برنامج DDE
٨٦٩	Object Linking and Embedding (OLE) ال
٨٧١	Word 95 OLE والنسخ الأقدم
٨٧٥	OLE object العينة لـ Word 97 وما أحدث
٨٨١	كتابة تطبيق حاوية OLE
٨٩٠	ملخص
٨٩٣	<b>الباب السابع عشر: تطوير تطبيقات قاعدة البيانات</b>
٨٩٤	Components
٨٩٧	تطوير قاعدة البيانات
٩٠٣	ال components ومصطلحات قاعدة البيانات
٩٠٩	ال Components قاعدة البيانات
٩٢٧	ال Structured Query Language
٩٢٨	إنشاء محرر SQL
٩٣٦	قواعد بيانات Master-Detail
٩٣٨	برمجة تطبيقات (master-detail)
٩٤٠	استخدام Data Modules
٩٤٦	برمجة قاعدة البيانات
٩٥١	الملخص
٩٥٣	<b>الباب الثامن عشر: تطوير ال Charts والتقارير</b>
٩٥٤	Components
٩٥٦	إنشاء ال Charts مع ال TeeChart
٩٨٤	إنشاء تقارير مع ال QuickReport
١٠٠٥	ملخص
١٠٠٧	<b>الجزء الرابع: تقنيات متقدمة</b>
١٠٠٩	<b>الباب التاسع عشر: التعامل مع ال exceptions</b>
١٠١٠	معلومات حول ال Exceptions
١٠٢٨	معالجة ال Exception
١٠٣١	الإشارة إلى حالة exception instance
١٠٣٣	الامساك بأنواع معينة من ال exception



\*\*\*\*\*

رقم الإيداع  
١١٣٥٧/٩٩ لسنة ١٩٩٩

الترقيم الدولي  
977-307-047-6